Michael Laube

Einstieg in SQL

CREATE INDEX
ORDER BY
WHERE
CREATE TABLE
SELECT * FROM
DELETE FROM
GROUP BY
CREATE VIEW

Für Ausbildung und Beruf

- Datenbanken und SQL – ohne Vorkenntnisse einsteigen
- Syntax, Abfragen und Datenmodellierung
- Mit Übungen, Musterlösungen und Praxistipps

Inkl. Übungsdatenbank zum Download

Rheinwerk Computing

Kapitel 1
Grundlagen kennenlernen und verstehen

Datenbanken und SQL haben in der Informatik die Aufgabe, Daten auf eine einfache Art und Weise zu verwalten. Zentrale Elemente sind hier die Abfragesprache SQL selbst, die Tabellen, in denen die Daten gespeichert werden und schließlich die Datenbanksysteme, in denen die Tabellen und Daten hinterlegt sind.

Sie haben eine gute Wahl getroffen, als Sie sich entschieden haben, die Datenbankabfragesprache *SQL* zu lernen!

Datenbanken, die die Abfragesprache SQL unterstützen, sind weltweit verbreitet. Sie werden in zahlreichen Unternehmen und Institutionen verwendet, um Daten bzw. Informationen zu strukturieren und effektiv zu verwalten. Denken Sie nur an Versicherungen, Banken, Behörden und viele weitere Institutionen, die darauf angewiesen sind, Informationen sicher und dauerhaft zu verwalten.

Vermutlich interessiert Sie auch, wofür SQL überhaupt steht:
Structured Query Language.

Structured Query Language (SQL)

S → Structured (engl. für »strukturiert«)

Q → Query (engl. für »Abfrage«)

L → Language (engl. für »Sprache«)

SQL ist eine *strukturierte Abfragesprache* für Datenbanken.

1.1 Die Tabelle als zentrales Element

Im Zentrum der Sprache SQL steht die *Tabelle*. Mit SQL können Sie folgende Hauptfunktionen auf eine Tabelle anwenden:

- Daten aus einer Tabelle *abfragen*
- Daten in eine Tabelle *einfügen*

- Daten in einer Tabelle *löschen*
- Daten in einer Tabelle *aktualisieren*

Beginnen werden Sie in diesem Buch mit sogenannten **SELECT**-Abfragen, die auf eine Tabelle angewendet werden können. Die in ihrer Grundstruktur einfachen Abfragen ermöglichen es Ihnen, Daten einer Tabelle abzurufen bzw. abzufragen. Außerdem hält SQL alles bereit, was nötig ist, um Daten in Tabellen einzufügen, zu ändern und natürlich auch zu löschen.

Sie merken schon, die Tabelle ist das zentrale Element von SQL.

1.1.1 Tabellen und ihre Struktur

Wahrscheinlich haben Sie bereits einmal in einem anderen Zusammenhang mit Tabellen zu tun gehabt, beispielsweise mit Microsoft Excel. Dann wissen Sie, dass die Struktur einer Tabelle grundsätzlich sehr einfach ist: Eine Tabelle besteht aus *Zeilen* und *Spalten*. Eine Zeile, die auch als *Datensatz* bezeichnet wird, enthält Werte, die wiederum Spalten zugeordnet sind.

Betrachten wir zur Illustration ein Beispiel. In Abbildung 1.1 sehen Sie die Tabelle, die von der Öffentlichkeit vermutlich am meisten diskutiert wird: die Tabelle der ersten Fußball-Bundesliga.

Pl.		Verein	Sp.	g.	u.	v.	Tore	Diff.	Pkte.
1	▲	» Borussia Dortmund	27	19	6	2	66:30	36	63
2	▼	» Bayern München (M)	27	19	4	4	69:28	41	61
3	–	» RB Leipzig	27	15	7	5	49:20	29	52
4	▲	» Eintracht Frankfurt (P)	27	14	7	6	54:30	24	49
5	▼	» Bor. Mönchengladbach	27	14	5	8	46:34	12	47
6	▲	» Werder Bremen	27	11	9	7	49:39	10	42
7	▼	» Bayer 04 Leverkusen	27	13	3	11	48:44	4	42

Abbildung 1.1 Diese Tabelle enthält die für viele Mitbürger essenziellen Daten zur schönsten Nebensache der Welt. (Quelle: https://www.kicker.de/news/fussball/bundesliga/spieltag/1-bundesliga/2018-19/27/0/spieltag.html)

Hier sehen Sie auf den ersten Blick, dass Borussia Dortmund am 27. Spieltag vor Bayern München steht. (Zugegeben: Dieser Stand ist nicht mehr taufrisch.) Die Informationen, die uns die Spalten der Tabelle liefern, sind:

- PL. – der Platz, auf dem sich Ihr Lieblingsclub befindet
- Verein – die Bezeichnung des Clubs
- Sp. – die Anzahl der gespielten Spiele
- G. – die Anzahl der Siege
- U. – die Anzahl der unentschieden gespielten Spiele

- V. – die Anzahl der Niederlagen
- Tore – die Anzahl der bisher erzielten Tore
- Diff. – die Tordifferenz
- Pkte. – die Anzahl der bis dato geholten Punkte

Die Spaltenbezeichnungen im Tabellenkopf, z. B. Pl., Verein, Sp. und G, stehen für die *Eigenschaften* der Tabelle. Sie sorgen dafür, dass der Analyst die Zahl der Spiele und die Zahl der Niederlagen auseinanderhalten kann. Darunter stehen die dazugehörigen Zeilen, die die Tabelle mit Inhalt füllen. Einen *Wert* in einer Tabelle ermitteln Sie, indem Sie den Schnittpunkt der Zeile und der Spalte betrachten. Sie wollen wissen, welcher Club auf Platz 1 ist? Kein Problem: Betrachten Sie Zeile 1 und die Spalte Verein, und ermitteln Sie den Schnittpunkt. Hier ist der Wert Borussia Dortmund zu finden.

Zusammenfassung: Definition einer Tabelle

Eine *Tabelle* besteht aus einem Tabellennamen, Zeilen und Spalten.

Der *Name* einer Tabelle fasst die Eigenschaften zusammen, die in einer Tabelle hinterlegt sind, und beschreibt, welche Informationen gespeichert werden.

Die *Eigenschaften* (Spalten) einer Tabelle bilden eine Struktur, in der Informationen geordnet gespeichert werden.

Die Informationen (die Daten) einer Tabelle sind in den Zeilen zu finden. Ein *Wert* innerhalb einer Zeile wird gelesen, indem der Schnittpunkt zwischen einer Zeile und einer Spalte ermittelt wird.

1.2 Eine kleine Historie von SQL

Eine Tabelle habe ich Ihnen bereits vorgestellt. Jetzt widmen wir uns der Frage, wer auf die Idee kam, Tabellen zur Speicherung großer Datenmengen zu nutzen, und auch die Grundlagen für SQL schuf.

Edgar F. Codd entwickelte in den 1970er-Jahren ein mathematisches Modell auf Grundlage der Mengenlehre. Die Struktur einer Tabelle wie der Bundesligatabelle kann auf dieser Grundlage beschrieben werden: Wir betrachten eine Menge. Diese Menge bezeichnen wir als *Bundesligatabelle* und statten sie mit Elementen wie *Platz* und *Verein* aus. Dies lässt sich in folgender Kurzschreibweise darstellen:

```
Bundesligatabelle={Pl,Verein,Sp,g,u,v,...}
```

Mathematisch betrachtet, bedeutet das nichts anderes als eine Definition der Struktur der Bundesligatabelle. In den geschweiften Klammern ist die Menge der Elemente (Spalten) aufgelistet, die für die Informationen der Bundesligatabelle erforderlich sind.

Mit der mathematischen Beschreibung einer Tabelle ist es in der Datenverarbeitung natürlich nicht getan. Denn das Ganze muss noch zum Leben erweckt werden.

Deshalb entwickelte Edgar F. Codd auch eine Sprache, mit der Daten basierend auf einfachsten Operationen abgefragt, eingefügt, gelöscht oder geändert werden können. Das Resultat dieser Überlegungen bildete schließlich die Grundlage für das *relationale Datenbankmodell* und die Sprache SQL. Die Arbeit von Edgar F. Codd wurde schließlich gekrönt, indem sie zu einem Standard erhoben wurde. Diese Standards werden heute von der *ISO (International Organization for Standardization)* festgelegt, erweitert und publiziert. In Abschnitt 1.4 erfahren Sie mehr über den SQL-Standard und seine Umsetzung durch unterschiedliche Datenbanksysteme.

1.3 Datenbanksysteme

Unterschiedlichste Anbieter haben zahlreiche Datenbanksysteme auf den Markt gebracht, die SQL unterstützen und Daten bzw. Informationen in Tabellen speichern. Datenbanksysteme, die Informationen in Tabellen speichern, werden als *relationale Datenbanksysteme* bezeichnet. Zu den bekanntesten Vertretern gehören:

- *IBM DB2*
- *Oracle DB*
- *Microsoft SQL Server*
- *Oracle MySQL*
- *MariaDB*
- *PostgreSQL*

Hierbei handelt es sich nur um eine kleine Auswahl von SQL-Datenbanken. Die Auswahl der Datenbanken habe ich bewusst getroffen, um Ihnen einerseits *kommerzielle Anbieter* und andererseits *Open-Source-Vertreter* von Datenbanksystemen vorstellen zu können.

Als führendes Datenbanksystem verwenden wir in diesem Buch *MySQL*. MySQL-Datenbanken sind sehr weit verbreitet. Das liegt unter anderem daran, dass die MySQL-Datenbank kostenlos angeboten und oft im Verbund mit der Web-Programmiersprache PHP verwendet wird. Außerdem steht Ihnen die MySQL-Datenbank für die wichtigsten Betriebssysteme (beispielsweise unterschiedliche Linux-Distributionen, Windows-Systeme, Apple macOS Sierra) zur Verfügung. Ein weiterer Vorteil ist die umfangreiche Dokumentation, die Sie jederzeit online unter *http://dev.mysql.com/doc* abfragen können.

Falls Sie stattdessen lieber mit einer *PostgreSQL*- oder einer *Microsoft SQL Server*-Datenbank lernen möchten, wird es Sie freuen zu erfahren, dass ich auf alle drei

genannten Datenbanken eingehe und Ihnen jeweils eine Übungsdatenbank zur Verfügung stelle.

Die Datenbankserver *PostgreSQL* und *Microsoft SQL Server* sind ebenfalls sehr gut dokumentiert. Die Dokumentationen finden Sie unter:

- *http://www.postgresql.org/docs*
- *https://msdn.microsoft.com/de-de/library/bb510741.aspx*

MariaDB

MariaDB ist ein recht neues Datenbanksystem, und es lohnt sich, einige Worte darüber zu verlieren. MariaDB ist ein sogenannter *Fork* von MySQL, also eine Abspaltung von diesem Projekt. Das MariaDB-Projekt bemüht sich, die volle Kompatibilität mit MySQL zu bewahren – die Unterschiede zwischen den beiden Datenbanken sind wirklich mit der Lupe zu suchen und betreffen hauptsächlich Lizenzfragen. Alle Erläuterungen, die Sie in diesem Buch zu MySQL finden, gelten daher ebenso für MariaDB. Sie finden das Projekt unter:

https://mariadb.org

1.4 SQL – ein Standard und seine Umsetzung

Der SQL-Standard der ISO wurde in der Vergangenheit als *ANSI-SQL-Standard* (ANSI: American National Standards Institute) bezeichnet. Der ISO-SQL-Standard soll auch international sicherstellen, dass Ihre Abfragen auf allen konformen Datenbanken zuverlässig funktionieren.

SQL: Ein Standard

Die Abfragesprache SQL ist im Standard ISO/IEC 9075-1:2016 festgelegt. Die Spezifikation kann unter *www.iso.org* gegen ein Entgelt bezogen werden.

Sie können davon ausgehen, dass unabhängig vom verwendeten Datenbanksystem der SQL-Sprachschatz zum größten Teil so umgesetzt wird, wie er im SQL-Standard beschrieben ist. Der kleinere Teil des Sprachumfangs, der nicht so realisiert wird, wie es der SQL-Standard vorsieht, wird als *SQL-Dialekt* bezeichnet. Alle drei hier behandelten Datenbanken haben einen eigenen Dialekt, der sich durch minimale Unterschiede zum SQL-Standard-Sprachumfang bemerkbar macht.

Aus diesem Grund stelle ich Ihnen hier SQL anhand von drei unterschiedlichen Datenbanksystemen vor. Einerseits möchte ich Ihnen damit zeigen, dass SQL ein Standard ist und sich daher das Gelernte auf unterschiedliche Datenbanken anwenden lässt. Andererseits möchte ich Sie dafür sensibilisieren, dass SQL in einem minimalen

Umfang abhängig vom jeweiligen Hersteller ist und damit im Detail unterschiedlich sein kann. Da es sich aber nur um kleine Unterschiede handelt, können Sie die hier vermittelten Grundlagen auch nutzen, um z. B. auf einer DB2-Datenbank von IBM oder einer Oracle-Datenbank mit SQL zu arbeiten.

[*]

Zusammenfassung: SQL-Dialekte

SQL ist ein Standard, der durch die ISO (International Organization for Standardization) vorgegeben wird. Dennoch gibt es im Sprachschatz jeweils abhängig von der verwendeten Datenbank Abweichungen vom Standard. Diese Unterschiede werden als *SQL-Dialekte* bezeichnet.

1.5 Zu diesem Buch

Grundsätzlich gehen wir in diesem Buch vom SQL-Standard aus. Den Grundsprachschatz, den Sie im Rahmen dieses Einstiegs in SQL erlernen, können Sie universal auf allen SQL-Datenbanken anwenden. Bezogen auf die hier behandelten Datenbanksysteme werde ich Sie stets auf die kleinen Unterschiede aufmerksam machen. Und wenn Sie einmal auf den Fall stoßen, dass ein hier beschriebenes Beispiel bei Ihnen nicht funktioniert, weil Sie ein besonderes Feature einer bestimmten Datenbank einsetzen wollen: Glückwunsch, dann sind Sie wahrscheinlich kein Einsteiger mehr und spezialisieren sich auf ein Datenbanksystem eines Anbieters. Achten Sie in diesem Fall besonders auf die Dokumentation der von Ihnen eingesetzten Datenbank.

In Abschnitt 1.6 gebe ich Ihnen einen Leitfaden an die Hand, der Sie bei der Installation der MySQL-Datenbank unterstützt.

Der Leitfaden zur Installation enthält ausschließlich Hinweise, die für die Installation der jeweiligen Datenbank von Bedeutung sind. Von einer gewissen Affinität zu Windows- und Linux-Systemen gehe ich in diesem Fachbuch aus. Unter Windows können Sie die hier vorgestellten Datenbanken sehr einfach installieren. Ausführlicher werde ich Ihnen die Installation einer MySQL-Datenbank unter Windows beschreiben.

In den Download-Materialien zum Buch finden sie das MySQL-Installationsprogramm in der folgenden Version hinterlegt:

- MySQL 8.0.16 Community Edition, MySQL Workbench 8.0.16

Falls Sie eine PostgreSQL- oder MS SQL Server-Datenbank verwenden möchten, laden Sie die Installationsprogramme bitte direkt beim Hersteller herunter. Die entsprechende Installationsanleitung finden Sie in den Materialien zum Buch auf *www.rheinwerk-verlag.de/4912*. Beachten Sie, dass sich je nach Version Abweichungen ergeben können. Die Erklärungen in diesem Buch beziehen sich auf die Versionen:

- PostgreSQL 11.2, pgAdmin 4
- MS SQL Server 2017, SQL Server Managementstudio

In Abschnitt 1.7 zeige ich Ihnen anschließend, wie Sie die Übungsdatenbank auf den hier besprochenen Systemen anlegen und die Daten importieren.

In Abschnitt 1.8 erfahren Sie, wie Sie die *MySQL Workbench* nutzen, um mit SQL zu beginnen.

Sie werden in diesem Buch viele Kästen und SQL-Beispiele finden. In Hinweiskästen befinden sich besonders wichtige Tipps, die Sie sich unbedingt anschauen sollten. Außerdem werde ich am Ende der Abschnitte die wichtigsten Informationen noch einmal für Sie zusammenfassen. Dies wird immer mit einem besonderen Icon gekennzeichnet. [*]

In den SQL-Listings finden Sie sowohl eine kürzere einzeilige Schreibweise als auch die etwas längere, aber übersichtlichere Fassung, die sich über mehrere Zeilen erstreckt:

```
SELECT
  name,
  vorname,
  bonus
FROM
  mitarbeiter
WHERE bonus>500;
```

Listing 1.1 Ein auf mehrere Zeilen verteilter SQL-Befehl

Für die Datenbank bedeutet diese Anweisung genau das Gleiche wie der folgende Befehl:

```
SELECT name,vorname,bonus FROM mitarbeiter WHERE bonus>500;
```

Listing 1.2 Das gleiche Beispiel in der Kurzschreibweise

Wenn Sie nur rasch eine SQL-Anweisung ausprobieren wollen, reicht Ihnen wahrscheinlich die Kurzschreibweise aus. Bei längeren, komplexeren Anweisungen wird dies jedoch schnell unübersichtlich. Dann ist es hilfreich, die einzelnen Schritte in der übersichtlichen Schreibweise aufzuschlüsseln. Dabei ist es sinnvoll, jedes SQL-Schlüsselwort auf eine Zeile zu schreiben. Egal ob in der Kurzschreibweise oder der ausführlichen Darstellung: Die SQL-Schlüsselwörter finden Sie hier im Buch immer fett gedruckt und durch Großbuchstaben hervorgehoben. So können Sie sich rasch einen Überblick über die Beispiele verschaffen.

Was diese Schlüsselwörter genau bedeuten, erfahren Sie in den folgenden Kapiteln. Kümmern wir uns aber zunächst darum, die MySQL-Datenbank zu installieren.

1.6 MySQL unter Windows installieren

Um MySQL unter Windows zu installieren, führen Sie die folgenden Schritte aus:

Schritt 1: Laden Sie entweder den MySQL-Installer der MySQL 8.0.16 Community Edition von der Webseite zu diesem Buch unter *www.rheinwerk-verlag.de/4912* oder die aktuelle Version von der folgenden Internetseite herunter: *https://dev.mysql.com/downloads/mysql*. Achten Sie dabei darauf, dass das Auswahlfeld unter dem Punkt SELECT OPERATION SYSTEM auf MICROSOFT WINDOWS eingestellt ist, um den MSI-Installer herunterladen zu können. Nach einem Klick auf GO TO DOWNLOAD PAGE werden Sie auf die Downloadseite geführt. Wählen Sie hier den MSI Installer zum Download aus – die Anmeldeinformation auf der folgenden Seite können Sie getrost ignorieren. Klicken Sie einfach weiter unten auf der Seite auf NO THANKS, JUST START MY DOWNLOAD, um die Datei ohne Anmeldung herunterzuladen.

Schritt 2: Öffnen Sie als Nächstes das Download-Verzeichnis, und führen Sie einen Doppelklick auf die soeben heruntergeladene Installationsdatei (mit der Endung *.msi*) aus. Falls Ihnen ein Dialogfenster mit einer Warnung angezeigt wird, in der Sie gefragt werden, ob Sie die Datei tatsächlich ausführen wollen, bestätigen Sie diese mit JA. Sollten während des Installationsprozesses weitere Warnmeldungen auftauchen, bestätigen Sie auch diese entsprechend.

Zunächst öffnet sich ein Fenster mit den Lizenzbedingungen (LICENSE AGREEMENT) wie in Abbildung 1.2, die leider nur auf Englisch vorliegen. Wenn Sie diese aufmerksam gelesen haben (und mit ihnen einverstanden sind), bestätigen Sie sie, indem Sie den Haken in der Checkbox I ACCEPT THE LICENSE TERMS setzen. Klicken Sie anschließend auf den NEXT-Button, um mit der Installation fortzufahren.

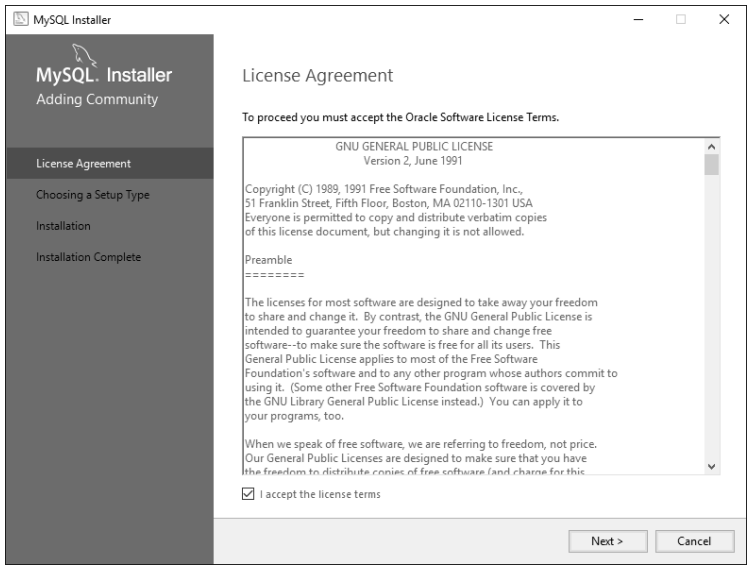


Abbildung 1.2 Die Lizenzbedingungen der MySQL-Datenbank

Schritt 3: Im Fenster CHOOSING A SETUP TYPE sehen Sie die fünf möglichen Installationsvarianten Abbildung 1.3. Wählen Sie die Variante CUSTOM. Diese ermöglicht, nur die Softwarekomponenten auszuwählen, die Sie für den Einstieg in SQL benötigen. Klicken Sie auf NEXT, um zum nächsten Installationsschritt zu gelangen.

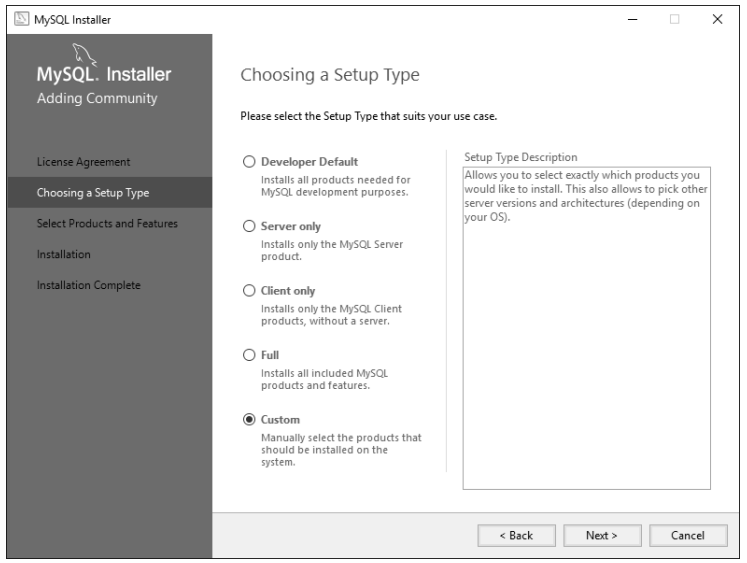


Abbildung 1.3 Installationstyp auswählen

Schritt 4: Im Fenster SELECT PRODUCTS AND FEATURES (siehe Abbildung 1.4) sehen Sie eine Auswahl der Softwarekomponenten, die Sie installieren können.

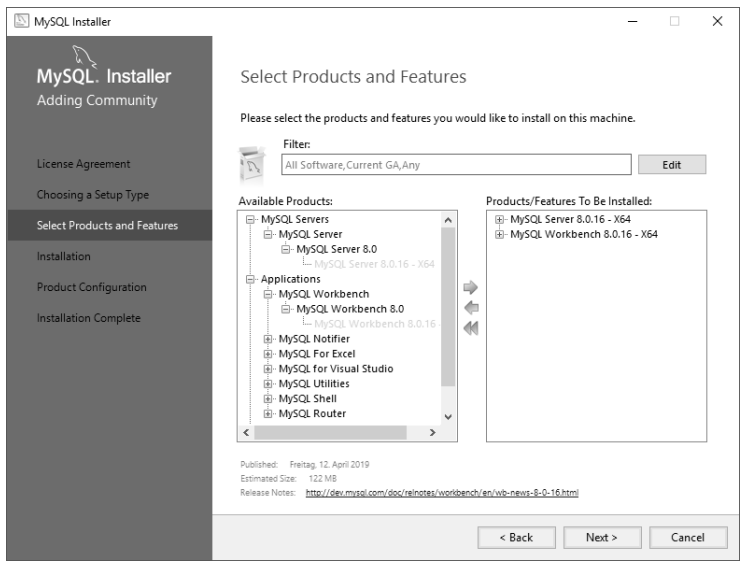


Abbildung 1.4 Zu installierende Komponenten auswählen

Im linken Fenster mit der Überschrift Available Products ist ein Navigationsbaum zu sehen. Klicken Sie einfach auf den Knoten mit dem Plus-Zeichen, um den Baum so weit aufzuklappen, bis die benötigten Softwarekomponenten auswählbar sind. Wählen Sie hier bitte MySQL Servers • MySQL Servers • MySQL Server 8.0 • MySQL Server 8.0.16–X64. Klicken Sie im Anschluss auf den grün gewordenen Pfeil nach rechts, der sich zwischen den beiden Fenstern befindet, um das Produkt in das Fenster Products/Features To Be Installed zu verschieben.

Wiederholen Sie diesen Schritt für Applications • MySQL Workbench • MySQL Workbench 8.0 • MySQL Workbench 8.0.16–X64.

Sollten neuere Versionen von MySQL Server und MySQL Workbench zur Verfügung stehen, wählen Sie bitte diese anstelle der hier genannten.

Klicken Sie anschließend auf den Button NEXT, um mit der Installation fortzufahren.

Schritt 5: Im Anschluss sehen Sie das Fenster Installation. In ihm werden nochmals die Softwareprodukte angezeigt, die Sie zur Installation ausgewählt haben. Mit einem Klick auf Execute werden diese heruntergeladen und installiert. Sobald die Installation erfolgreich war, werden vor den Softwarekomponenten grüne Häkchen angezeigt, und die Status-Spalte zeigt Complete an (siehe Abbildung 1.5). Klicken Sie erneut auf den Button Next, um zum nächsten Schritt zu gelangen.

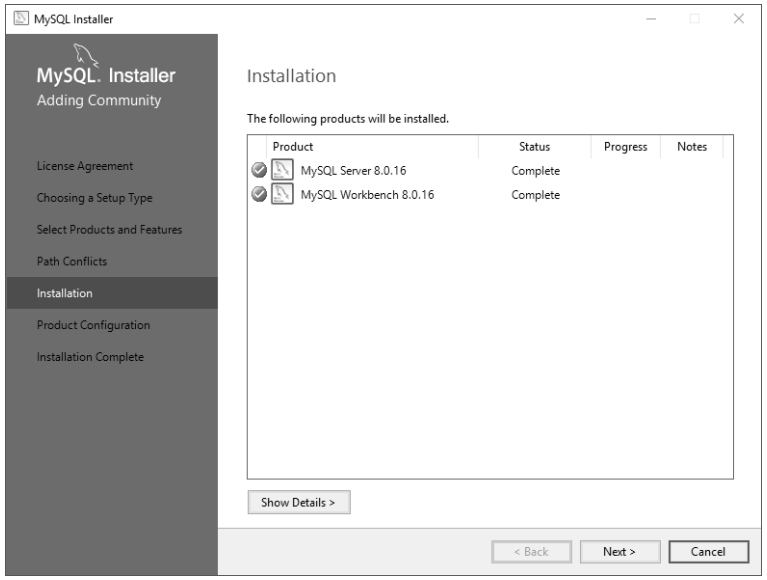


Abbildung 1.5 Die Installation der MySQL Server- und MySQL Workbench-Software wird bestätigt.

Schritt 6: Jetzt befinden Sie sich im Fenster Product Configuration. Hier werden die Softwareprodukte aufgelistet, die zur Konfiguration bereitstehen. Da Sie nichts weiter tun können, klicken Sie direkt auf Next.

Schritt 7: In dem folgenden Fenster Group Replication (siehe Abbildung 1.6) ändern Sie bitte keine Einstellung, da die vorausgewählte Variante Standalone MySQL Server / Classic MySQL Replication bereits die richtige Wahl darstellt. Klicken Sie auch hier auf Next, um zum nächsten Installationsschritt zu gelangen.

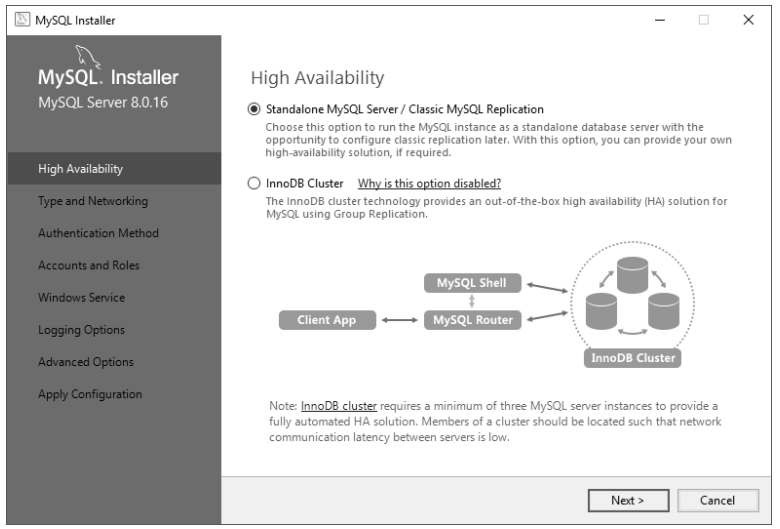


Abbildung 1.6 Group Replication

Schritt 8: Nach einem erneuten Klick auf Next öffnet sich das Fenster Type and Networking (siehe Abbildung 1.7). Der Konfigurationsdialog enthält einige voreingestellte Netzwerkeinstellungen. Ich empfehle Ihnen, diese beizubehalten und direkt auf Next zu klicken, um fortzufahren.

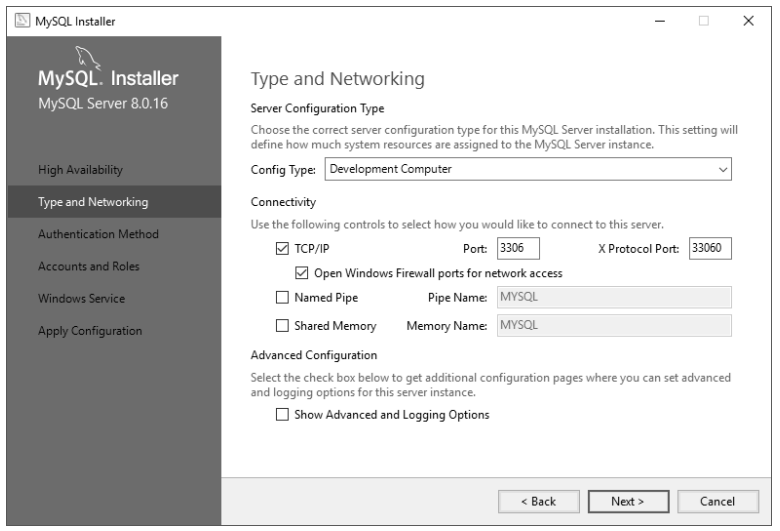


Abbildung 1.7 Typ und Netzwerkeinstellungen konfigurieren

Schritt 9: Im Fenster AUTHENTICATION METHOD (siehe Abbildung 1.8) können Sie zwischen zwei Optionen wählen. Belassen Sie es auch hier bei der Standardauswahl, und klicken Sie auf den Button NEXT, um die Installation fortzuführen.

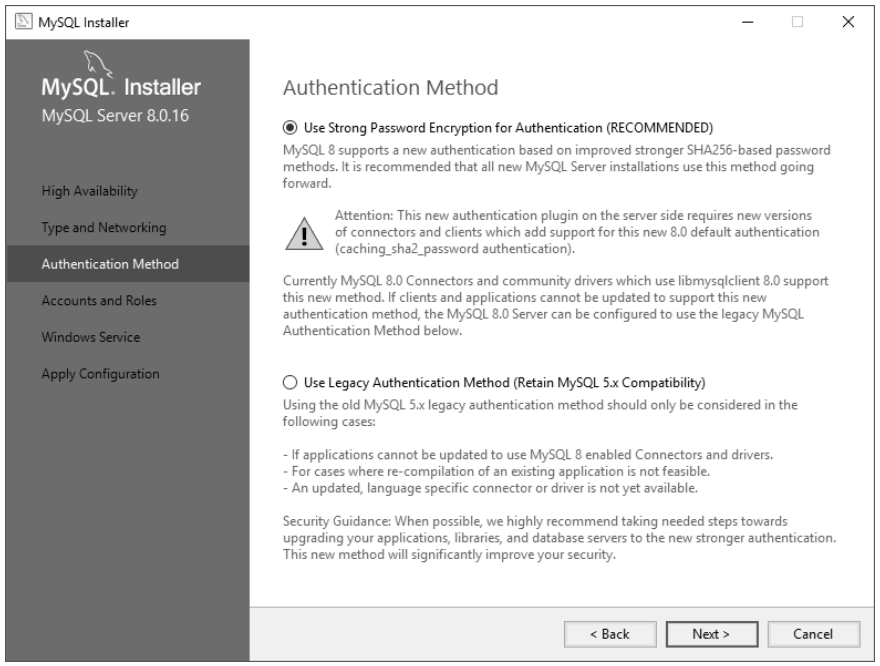


Abbildung 1.8 Die Authentifizierungsmethode auswählen

Schritt 10: Jetzt muss noch ein Passwort für unseren Superuser (*root*) festgelegt werden. Geben Sie ein Passwort Ihrer Wahl in das Texteingabefeld für MYSQL ROOT PASSWORD ein, und bestätigen Sie dieses noch einmal, indem Sie es erneut in das Texteingabefeld REPEAT PASSWORD eingeben.

Der Einfachheit halber empfehle ich Ihnen für den Übungsbetrieb, hier ein simples Passwort auszuwählen. Klicken Sie wieder auf den Button NEXT, um zum nächsten Installationsschritt zu gelangen.

Hinweis zur Nutzung des Administrators »root«

Im Produktivbetrieb einer Datenbank würden Sie aus Sicherheitsgründen natürlich niemals mit dem administrativen Nutzer `root` arbeiten, wenn Sie SQL-Abfragen durchführen. Für den Übungsbetrieb können Sie jedoch getrost den Administrator `root` verwenden. Mehr zu Benutzern, Rollen und Berechtigungen erfahren Sie in Kapitel 6.

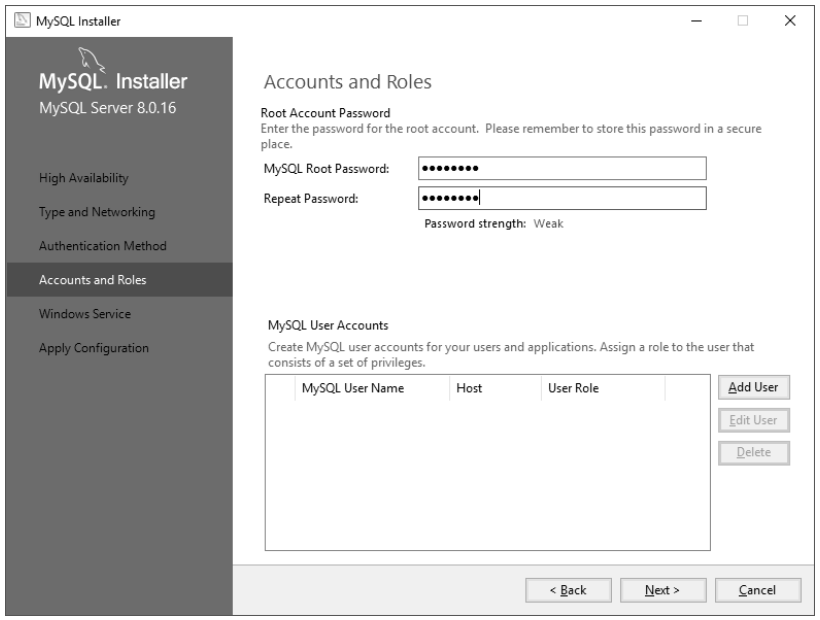


Abbildung 1.9 Ein Passwort für den Superuser (root) vergeben

Schritt 11: In Abbildung 1.10 sehen Sie das Konfigurationsfenster WINDOWS SERVICE. Der MySQL-Datenbankserver schlägt hier vor, sich als Windows-Dienst einzurichten. Idealerweise belassen Sie es dabei und klicken auf den Button NEXT, um fortzufahren.

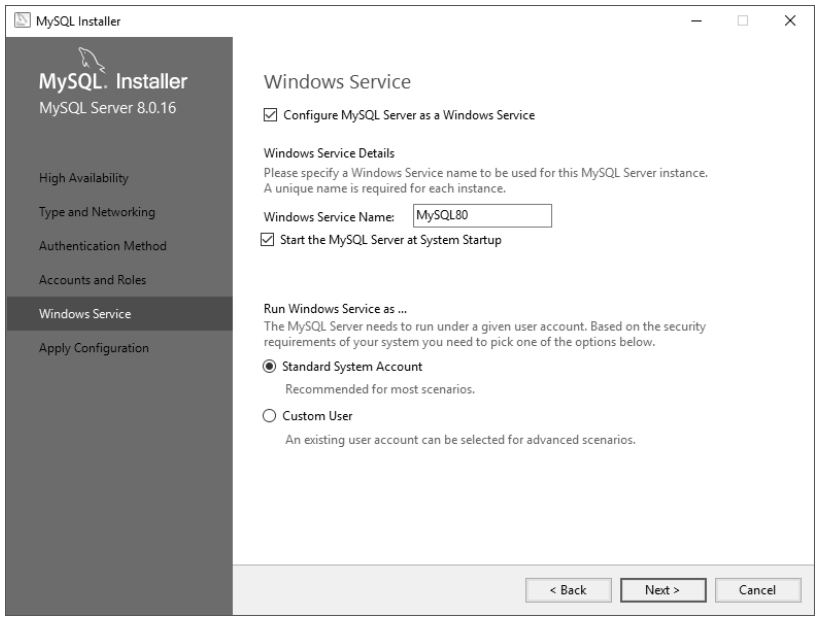


Abbildung 1.10 MySQL Server als Windows-Dienst einrichten

Schritt 12: Im Fenster APPLY CONFIGURATION müssen Sie mit einem Klick auf EXECUTE nur noch der Konfiguration zustimmen.

Gleich im Anschluss bestätigt Ihnen das Installationsprogramm mit grünen Häkchen, welche Konfigurationsschritte bereits ausgeführt worden sind (siehe Abbildung 1.11). Sobald alle Schritte durchgeführt wurden, können Sie auf FINISH klicken, um die Konfiguration abzuschließen.

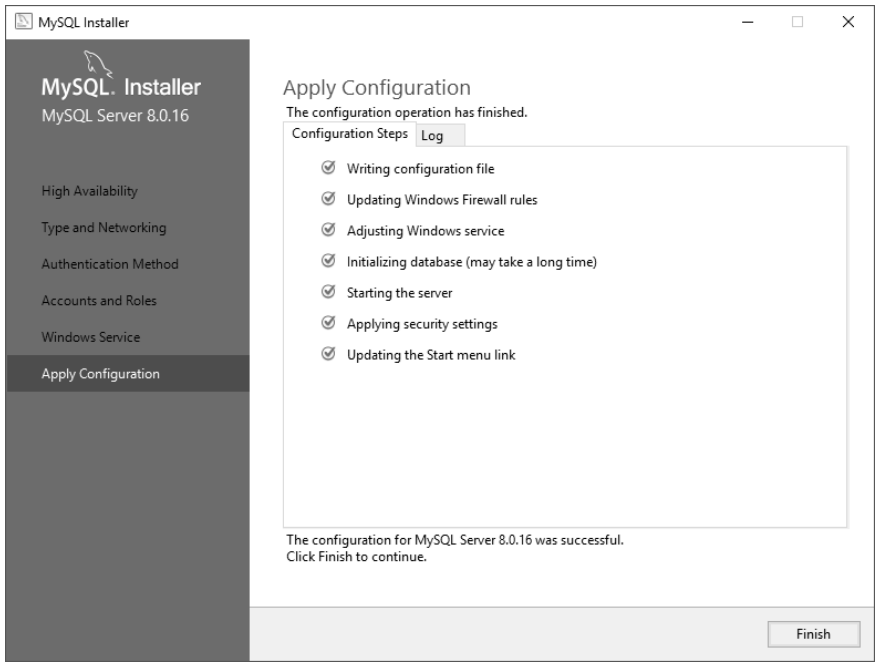


Abbildung 1.11 Bestätigung der Konfiguration

Abschließend erhalten Sie eine Bestätigung über die erfolgreiche Installation und Konfiguration von MySQL Server. Klicken Sie abermals auf NEXT.

Jetzt haben Sie es endlich geschafft! Die MySQL Server-Datenbank und die MySQL Workbench wurden installiert und sämtliche erforderlichen Einstellungen wurden vorgenommen.

Aktivieren Sie, falls es noch nicht ausgewählt sein sollte, im nachfolgenden Fenster das Kontrollkästchen START MYSQL WORKBENCH AFTER SETUP (siehe Abbildung 1.12). Nun müssen Sie nur noch auf FINISH klicken, um die Installation abzuschließen und MySQL Workbench zu starten.

Im folgenden Willkommensfenster der MYSQL WORKBENCH sollten Sie darauf achten, im linken Navigationsbereich nur das Symbol mit dem Delphin auszuwählen (siehe Abbildung 1.13).

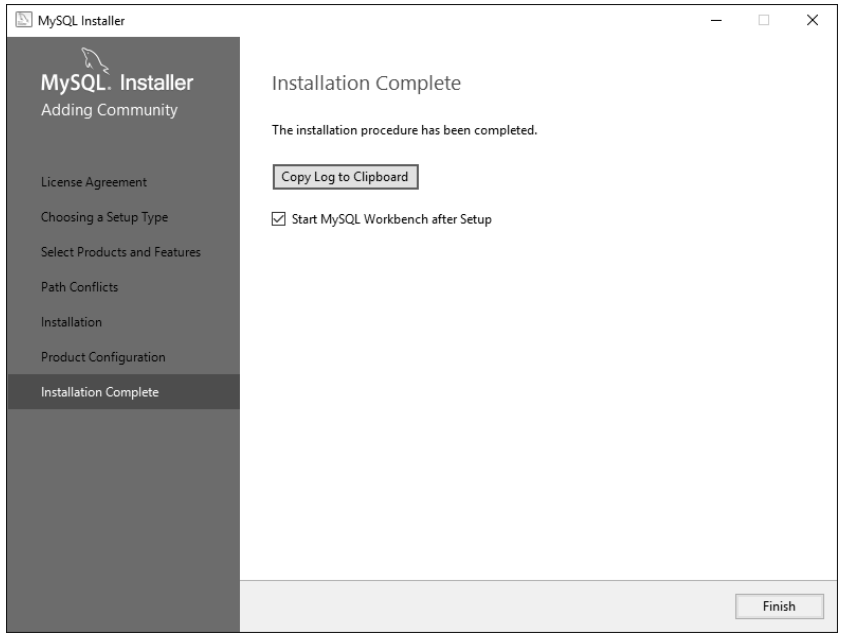


Abbildung 1.12 Abschluss der Installation

Im Hauptfenster wird Ihnen eine grau hinterlegte Fläche mit der Bezeichnung LOCAL INSTANCE angezeigt, auf die eine MySQL-Versionsbezeichnung folgt. Wenn Sie auf diese Fläche klicken, öffnet sich nach der Eingabe des Passworts für den Benutzer *root* die Editoransicht. Sie ist im Prinzip ein Texteditor, in dem Sie SQL-Anweisungen eingeben und ausführen können.

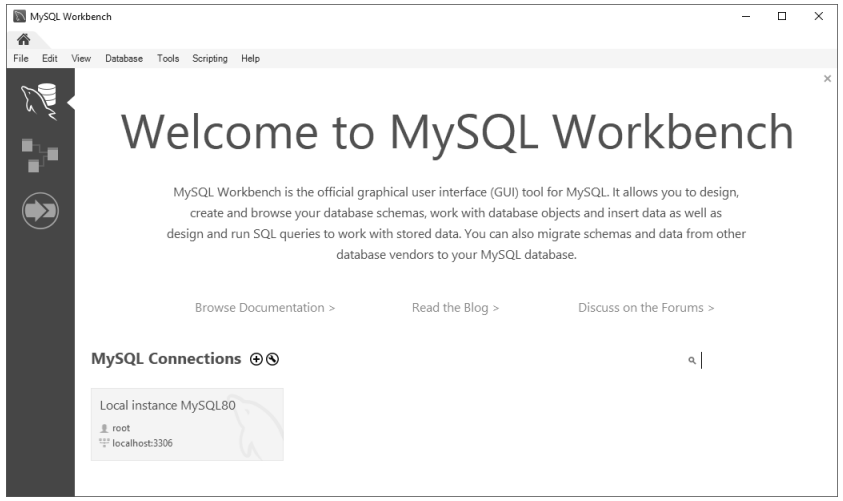


Abbildung 1.13 Die MySQL Workbench nach dem Start

1.7 Die MySQL-Übungsdatenbank anlegen

An dieser Stelle gehe ich davon aus, dass Sie, wie in Abschnitt 1.6 beschrieben, die MySQL-Datenbank installiert haben.

Die Übungsdatenbank trägt den selbsterklärenden Namen *uebungsdatenbank*. Beachten Sie, dass der Name vollständig kleingeschrieben ist. Der deutsche Umlaut »ü« wurde durch die Buchstaben »ue« ersetzt, da sich ansonsten Probleme mit Zeichensätzen ergeben können. Daher gehört es in der Programmierung zum guten Stil, Bezeichner stets ohne Umlaute zu verwenden.

Eine wichtige Bemerkung vorab: Um die Übungsdatenbank anzulegen, benötigen Sie *Administratorrechte* für die verwendete Datenbank. Wenn Sie der Installationsanleitung in Abschnitt 1.6 gefolgt sind, besitzen Sie bereits Administratorrechte. Während der Installation haben Sie für den Benutzer `root` ein Passwort vergeben.

Administratoren unter MySQL

Unter MySQL heißt der Nutzer, der über administrative Rechte verfügt, `root`.

Um die Übungsdatenbank erfolgreich anzulegen, benötigen Sie Administratorrechte für die Datenbank oder entsprechende Berechtigungen, die es Ihnen erlauben, Datenbanken oder Tabellen anzulegen und natürlich Daten dort einzufügen.

Kommen wir jetzt zu den Vorbereitungen, die für das Importieren der Übungsdatenbank notwendig sind. Laden Sie die Importskripte für die Übungsdatenbanken herunter. Beachten Sie, dass die Downloaddatei die Importskripte für alle drei Datenbanken enthält, die hier vorgestellt werden.

Download der Importskripte

Sie finden die Importskripte auf der Webseite des Rheinwerk Verlags zu diesem Buch unter *www.rheinwerk-verlag.de/4912*.

Wählen Sie dort den Tab mit der Bezeichnung Materialien zum Buch. Das bzw. die Importskripte stehen als *ZIP*-Datei zum Download bereit. Bitte laden und entpacken Sie diese Datei. Das Entpacken der ZIP-Datei kann beispielsweise in Ihrem *Downloads*-Ordner erfolgen.

In dem soeben entpackten Verzeichnis *mysql* finden Sie die folgenden Dateien:

- *importskript.sql*
- *importskript_arbeitszeit.sql*
- *importskript_kreditinstitut.sql*

Wir benötigen zunächst nur die Datei *importskript.sql*, mit der wir die Übungsdatenbank anlegen.

Abbildung 1.14 zeigt das geöffnete Fenster der MySQL Workbench. Klicken Sie auf das Ordnersymbol unterhalb des Reiters (hier SQL File 3), um den Dateiauswahldialog Open SQL Scripts zu öffnen.

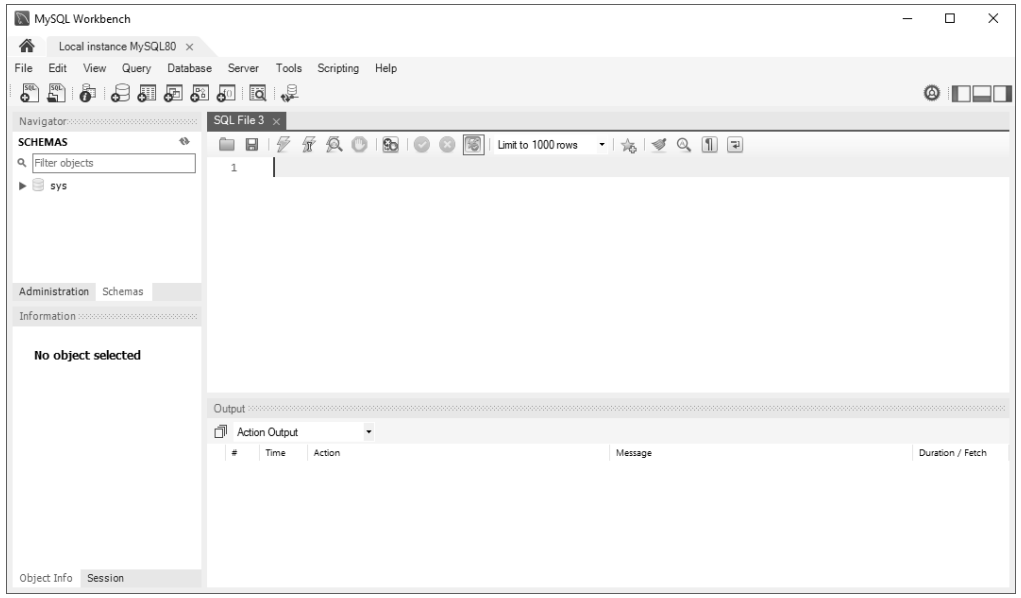


Abbildung 1.14 Der Editor der MySQL Workbench

Wählen Sie hier im Ordner *Downloads\mysql* die Datei *importskript.sql* aus (siehe Abbildung 1.15), und öffnen Sie sie mit einem Klick auf Öffnen.

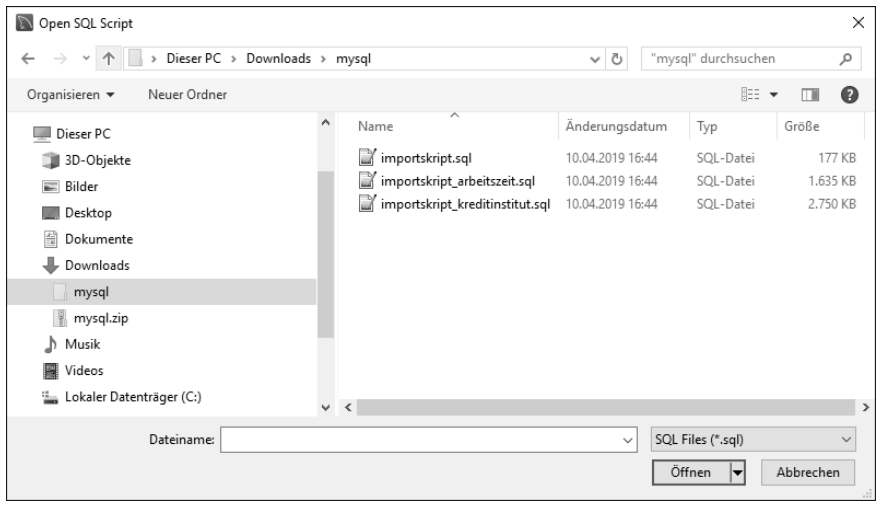


Abbildung 1.15 Das Importskript auswählen und öffnen

Stellen Sie nun sicher, dass sich der Cursor im geöffneten Importskript an der ersten Position der ersten Zeile befindet (siehe Abbildung 1.16). Klicken Sie danach auf den *gelben Blitz*, um alle Anweisungen, die im Importskript enthalten sind, auszuführen und die Übungsdatenbank anzulegen.

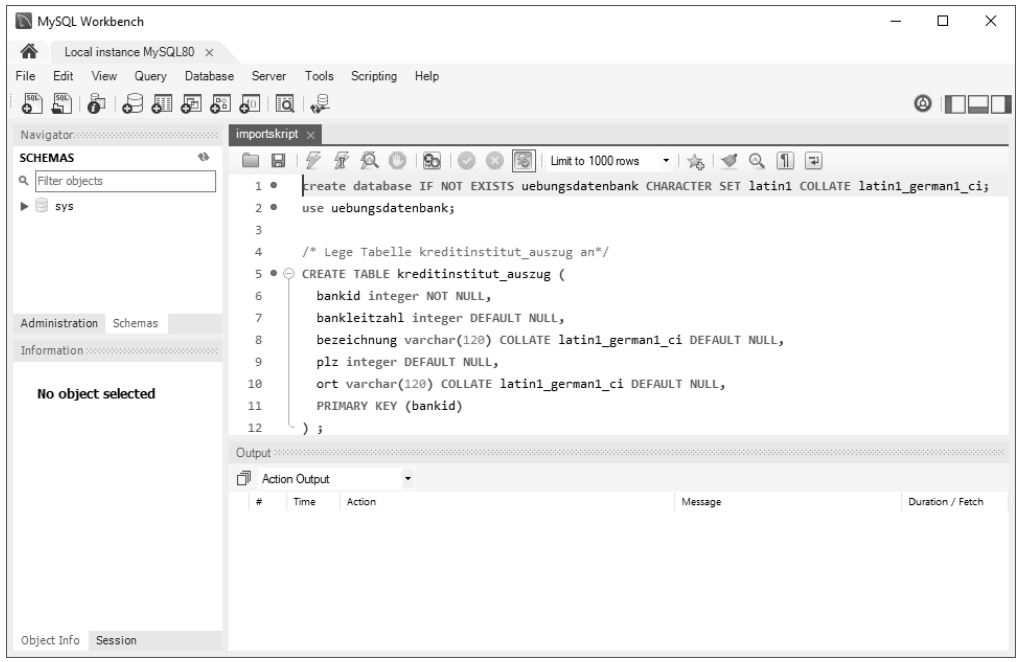


Abbildung 1.16 Das geöffnete Importskript in der MySQL Workbench

In Abbildung 1.17 sehen Sie im Fenster ACTION OUTPUT, dass die Anweisungen, die für den Import erforderlich sind, ausgeführt wurden.

Nun müssen wir die Liste der Datenbanken aktualisieren. Diese werden im MySQL-Sprachgebrauch auch *Schema* genannt. Klicken Sie hierfür auf den Reiter SCHEMAS, der sich unterhalb des NAVIGATOR-Fensters befindet. Nun sehen Sie im NAVIGATOR-Fenster rechts neben der Überschrift SCHEMAS ein Aktualisierungssymbol. Wenn Sie auf dieses klicken, wird Ihnen die importierte Datenbank mit ihren Tabellen angezeigt.

Die aktive Datenbank wird im NAVIGATOR-Fenster wie in Abbildung 1.18 immer in Fettschrift dargestellt. Stellen Sie zu Beginn Ihrer Arbeit stets sicher, dass Sie im NAVIGATOR-Fenster auf die *uebungsdatenbank* geklickt haben und diese fettgedruckt angezeigt wird.

Wenn Sie zu einem späteren Zeitpunkt mit mehreren Datenbanken auf Ihrem MySQL-Datenbanksystem arbeiten, können Sie mit der MySQL Workbench auch auf diese entsprechend zugreifen.

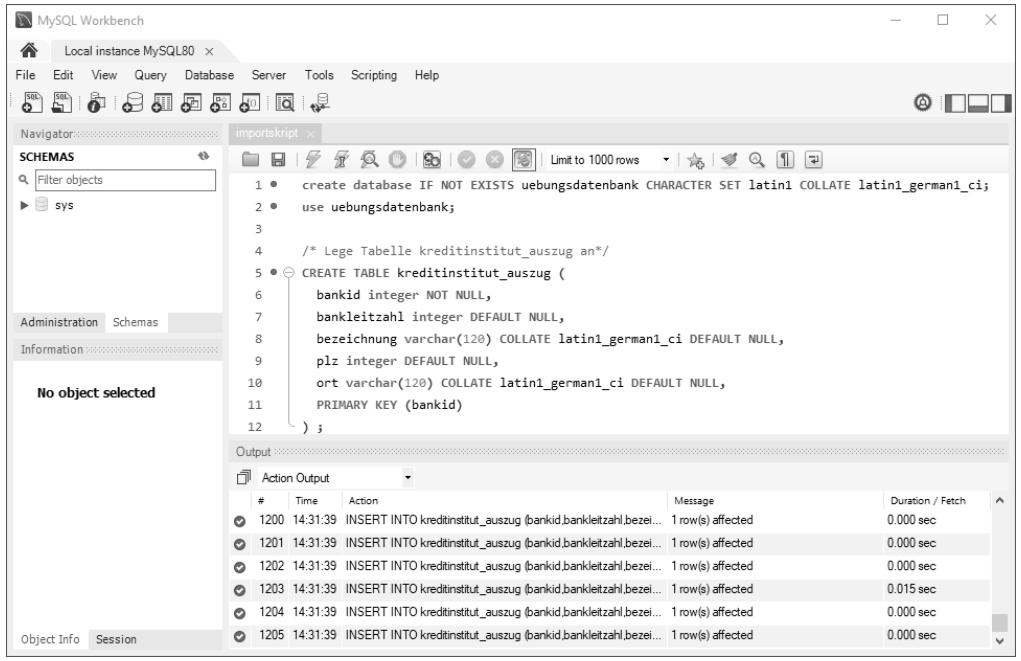


Abbildung 1.17 Das Importskript wurde ausgeführt.

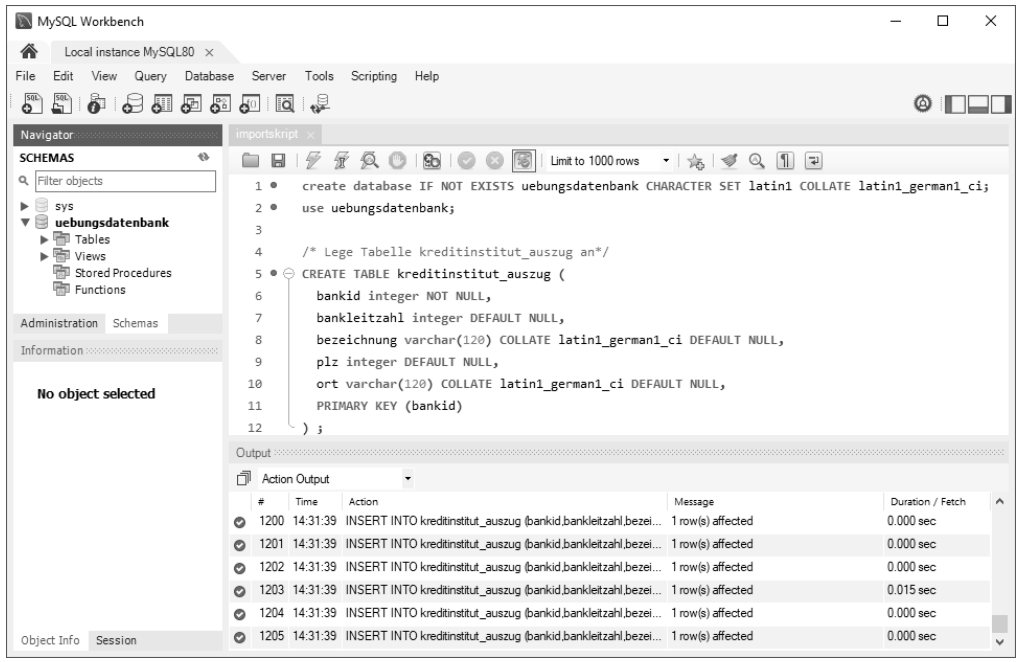


Abbildung 1.18 Das Navigator-Fenster wurde aktualisiert.

Schließen Sie nun das geöffnete Importskript, indem Sie auf das Kreuz neben dem Tab IMPORTSKRIPT klicken. Alternativ können Sie auch die Tastenkombination Strg + W nutzen, um das Editorfenster zu schließen.

1.8 Eine erste Abfrage an die Datenbank senden

Wir werden nun eine erste einfache Abfrage der Tabelle *mitarbeiter* ausführen. Da wir in Abschnitt 1.7 das Editorfenster geschlossen haben, müssen Sie ein neues Editorfenster in der MySQL Workbench aufrufen. In Abbildung 1.19 sehen Sie die Menüleiste der MySQL Workbench. Unter den Menüpunkten sehen Sie ein paar Icons:

❶ Das erste Icon von links enthält ein Pluszeichen. Sie verwenden es, wenn Sie beabsichtigen, eine neue Datei zu erstellen, in der Sie Ihre SQL-Anweisungen notieren und ausführen.

❷ Das zweite Icon zeigt eine geöffnete Mappe. Hiermit können Sie eine vorhandene SQL-Datei öffnen. Sie haben es bereits kennengelernt, als wir das Importskript geöffnet haben.

❸ Das dritte Icon enthält ein »i« wie Information. Wenn Sie z. B. im Navigationsbaum eine Datenbank markieren und auf dieses Icon klicken, so erhalten Sie ausführliche Informationen wie die Dateigröße oder enthaltene Tabellen zu Ihrer Datenbank.

❹ Das vierte Icon symbolisiert eine Datenbank. Mit ihm können Sie eine neue Datenbank (ein neues Schema) anlegen.

❺ Das fünfte Symbol zeigt eine Tabelle mit Pluszeichen. Mit ihm können Sie eine neue Tabelle für eine ausgewählte Datenbank anlegen.

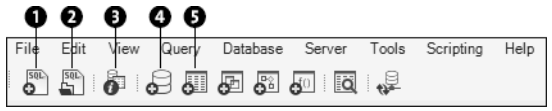


Abbildung 1.19 Die Menüleiste der MySQL Workbench

Nun wollen wir eine SQL-Abfrage an die Datenbank richten. Hierzu benötigen wir ein neues Editorfenster, das Sie mit einem Klick auf das erste Icon erzeugen. Alternativ können Sie auch das Tastaturkürzel Strg + T verwenden.

Am oberen Rand des Editorfensters sehen Sie nun wie in Abbildung 1.20 eine weitere Leiste mit Icons.

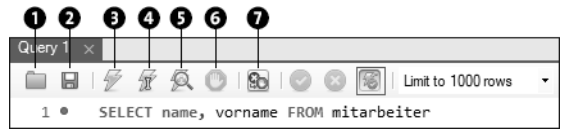


Abbildung 1.20 Eine SELECT-Abfrage ausführen

❶ Mit dem Ordnersymbol-Icon öffnen Sie eine gespeicherte *.sql*-Datei, in der Sie SQL-Anweisungen notiert haben.

❷ Mit dem Disketten-Icon speichern Sie Ihre aktuell bearbeiteten SQL-Anweisungen in einer Datei mit der Endung *.sql*.

❸ Das Blitz-Icon dient zur Ausführung von SQL-Anweisungen. Diese Version des Blitzes führt alle SQL-Anweisungen aus, die im SQL-Editor notiert wurden, es sei denn, Sie haben explizit eine Anweisung mit der Maus markiert.

❹ Das zweite Blitz-Icon (mit einem Cursor-Symbol) führt die SQL-Anweisung aus, die sich unterhalb des Cursors befindet.

❺ Das Blitz-Icon mit einer Lupe führt die SQL-Anweisung aus und liefert Ihnen eine Beschreibung dessen, was die Datenbank mit dieser Anweisung an Arbeitspaketen zu bewältigen hat. Dieser Punkt ist für Sie im Augenblick nicht von Bedeutung.

❻ Das Symbol mit der Hand bricht eine länger andauernde SQL-Anweisung ab. Diese Funktion werden Sie erst einmal nicht benötigen. Sie werden sie aber zu schätzen lernen, wenn Sie mit großen Datenvolumen zu tun haben.

❼ Über das Symbol eines Abbruchzeichens mit einer stoppenden Hand bestimmen Sie, ob SQL-Anweisungen weiter ausgeführt werden sollen, wenn eine SQL-Anweisung einen Fehler während der Ausführung zurückliefert.

Steigen wir gleich ein, und probieren wir eine Anweisung in einem Beispiel aus. Geben Sie dazu die SQL-Anweisung aus Listing 1.3 in Ihren Editor ein. Sie müssen an dieser Stelle noch nicht verstehen, was diese **SELECT**-Anweisung bewirkt, denn es kommt erst einmal nur darauf an, wie Sie SQL-Anweisungen eingeben und ausführen, um ein Ergebnis von der Datenbank zu erhalten.

```
SELECT name,vorname FROM mitarbeiter;
```

Listing 1.3 Eine SQL-Anweisung an die Datenbank richten

Jetzt müssen wir die im SQL-Editor notierte SQL-Anweisung nur noch an die Datenbank senden, um ein Ergebnis zu erhalten. Hierzu nutzen Sie die Iconleiste über dem Eingabeeditor, die Sie ja bereits kennengelernt haben. Nutzen Sie das Blitz-Icon ❸ aus Abbildung 1.20, um die SQL-Anweisung, die im SQL-Editor notiert ist, an die Datenbank zu richten. Anstelle des Symbols können Sie alternativ auch eine Tastenkombination verwenden, um Ihre SQL-Anweisungen an das MySQL-Datenbanksystem zu

senden: Halten Sie die Taste Strg gedrückt, und betätigen Sie die ↵-Taste, um die SQL-Anweisung im Editor auszuführen.

Führen Sie also Ihre erste SQL-Anweisung aus, indem Sie sie an den MySQL-Datenbankserver senden. Sie erhalten dann von der Datenbank eine Ergebnistabelle zurück (siehe Abbildung 1.21).

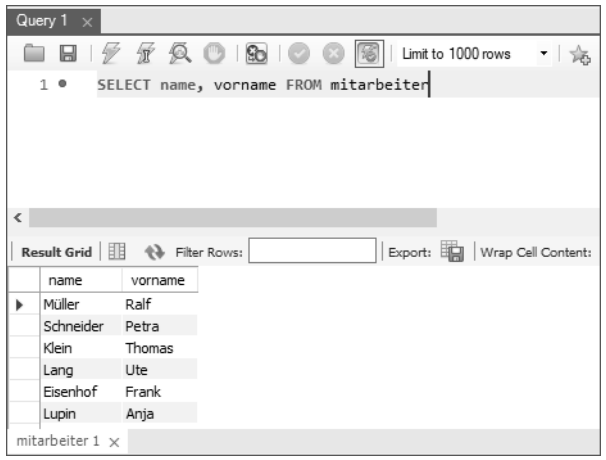


Abbildung 1.21 Ergebnistabelle nach der Ausführung einer SQL-Anweisung

Unterhalb der Ergebnistabelle sehen Sie ein Ausgabefenster wie Abbildung 1.22. In diesem Fenster wird Ihnen das Feedback der Datenbank zu den jeweiligen SQL-Anweisungen mitgeteilt. Der Haken in der ersten Spalte zeigt Ihnen, dass die jeweilige SQL-Anweisung erfolgreich ausgeführt wurde. Insgesamt hält das Ausgabefenster folgende Informationen für Sie bereit:

- den Zeitpunkt der Ausführung (*Time*)
- die SQL-Anweisung, die ausgeführt wurde (*Action*)
- eine Nachricht, die z. B. ausgibt, wie viele Zeilen zurückgeliefert wurden (*Message*)
- (nicht in der Abbildung enthalten) die Dauer, die zur Ausführung der SQL-Anweisung benötigt wurde

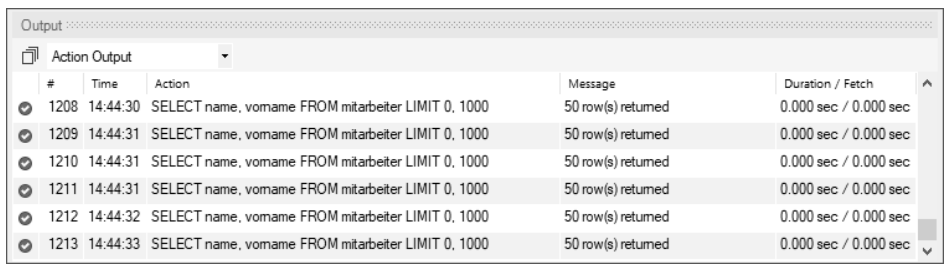


Abbildung 1.22 Statusmeldungen zu den SQL-Anweisungen

Gebräuchliche Tastaturkürzel der MySQL Workbench

Ich verwende gern Tastaturkürzel, da sie ein effektiveres Arbeiten ermöglichen. Aus diesem Grund stelle ich Ihnen die hier verwendeten Tastaturkürzel kurz zusammengefasst vor:

Strg + ⇧ + O – ein bestehendes Skript öffnen

Strg + T – einen neuen Tabulator (Editorfenster) öffnen

Strg + S – ein Skript speichern

Strg + W – ein Editorfenster schließen

Strg + ↵ – eine SQL-Anweisung ausführen

1.9 Kommentarfunktion

Sie sollten Ihre SQL-Anweisungen, die Sie dauerhaft speichern möchten, ausreichend mit erklärenden Notizen versehen. Besonders bei komplexeren SQL-Anweisungen ist das sehr hilfreich: Nur so können Sie auch nach längerer Zeit noch nachvollziehen, wie Sie die Anweisung programmiert und auf welchen Wegen Sie das Ergebnis herbeigeführt haben.

Ihre Kommentare können dabei über verschiedenste Dinge informieren. Als Beispiel liste ich Ihnen einige Informationen auf, die Ihren SQL-Code bereichern und Ihnen in Zukunft die Arbeit damit vereinfachen werden:

- die anfordernde Stelle
- die Funktion der Abfrage
- das Datum der Fertigstellung
- die Durchführung von Änderungen und der Grund der Änderung
- der Name des Programmierers

Grundsätzlich verfügt SQL über zwei unterschiedliche Arten von Kommentaren. So gibt es einzeilige Kommentare, die mit einem doppelten Bindestrich »`--`« eingeleitet werden, auf den der Text des Kommentars folgt. Daneben gibt es Kommentare, die sich über mehrere Zeilen erstrecken können. Diese Kommentare beginnen mit den Zeichen `/*` und enden mit den Zeichen `*/`. Alles, was dazwischen steht, wird vom Datenbanksystem ignoriert.

1.9.1 Kommentare in der Praxis nutzen

Sehen wir uns an, wie Sie einen einzeiligen Kommentar in der Praxis verwenden. Hierzu setzen wir einfach vor der `SELECT`-Abfrage einen einzeiligen Kommentar wie

in Listing 1.4 ein. Wenn Sie die Anweisung in Ihrem SQL-Client ausführen, ignoriert das Datenbanksystem den Kommentar und liefert Ihnen stattdessen nur das Ergebnis der **SELECT**-Abfrage als Ergebnistabelle zurück.

```
-- Spaltenauswahl: name, vorname der Tabelle mitarbeiter
SELECT name,vorname FROM mitarbeiter;
```

Listing 1.4 Einen einzeiligen Kommentar im SQL-Code anwenden

Einzeilige Kommentare sind zu Dokumentationszwecken nur eingeschränkt verwendbar, da Ihnen nur eine Zeile zur Verfügung steht. Aber dafür gibt es ja die mehrzeiligen Kommentare. Die nächste **SELECT**-Anweisung dokumentieren wir in Listing 1.5 mit einem mehrzeiligen Kommentar, um sie mit verschiedenen Informationen anzureichern. Erläutert wird so, wer der Autor war, wann der Befehl erstellt wurde, und es folgen noch einige weitere nützliche Informationen.

```
/*
Autor: Michael Laube
Erstellungsdatum: 01.08.2016
Funktion: Liste mit den Namen und Vornamen der Mitarbeiter
Funktion geprüft: ja
Funktion erfüllt: ja
Nutzergruppen der Abfrage: Personalabteilung
*/
SELECT name,vorname FROM mitarbeiter;
```

Listing 1.5 Einen mehrzeiligen Kommentar im SQL-Code anwenden

Die Datenbank ignoriert bei der Ausführung der SQL-Anweisung auch den mehrzeiligen Kommentar. Wenn Sie den gesamten Inhalt einer *.sql*-Datei auskommentieren, passiert einfach gar nichts. Bei der Ausführung der leeren Anweisung wird Ihnen kein Ergebnis zurückgeliefert, da keine SQL-Anweisung vorhanden ist.

[*]

Zusammenfassung: SQL-Anweisungen kommentieren

Kommentare werden zu Dokumentationszwecken verwendet. In SQL haben Sie zwei Möglichkeiten, Kommentare zu Ihren SQL-Anweisungen zu verfassen:

- Die erste Möglichkeit besteht darin, einzeilige Kommentare zu verfassen, die mit einem doppelten Bindestrich (--) eingeleitet werden.
- Die zweite Möglichkeit ist, mehrzeilige Kommentare zu verfassen. Diese beginnen stets mit einem Slash, gefolgt von einem Sternchen (/*), und enden mit einem Sternchen, gefolgt von einem Slash (*/).

1.9.2 Übungen

Übung 1

Geben Sie die SQL-Anweisung wie in Listing 1.6 gezeigt in Ihre MySQL-Workbench ein. Vor der **SELECT**-Anweisung soll folgender einzeiliger Kommentar notiert werden:

Alle Spalten der Tabelle mitarbeiter abfragen

```
SELECT * FROM mitarbeiter;
```

Listing 1.6 Alle Spalten einer Tabelle abfragen

Führen Sie die Abfrage in Ihrem SQL-Client aus, um die Auswirkungen des einzeiligen Kommentars auf das Ergebnis der Abfrage zu ermitteln.

Übung 2

In dieser Übung sollen sämtliche Mitarbeiter ermittelt werden, die zur Abteilung *einkauf* gehören. Bitte schreiben Sie die SQL-Anweisung wie in Listing 1.7 angegeben in Ihr Editorfenster:

```
SELECT
  name,vorname,abteilung
FROM
  mitarbeiter
WHERE
  abteilung='einkauf';
```

Listing 1.7 Eine weitere SQL-Abfrage, die mit einem mehrzeiligen Kommentar versehen werden muss

Vor der **SELECT**-Anweisung notieren Sie einen Kommentar, der so strukturiert ist:

Programmierer: Ihr Name

Datum: Das aktuelle Datum

Funktion: Filtern aller Mitarbeiter, die zur Abteilung Einkauf gehören

Beachten Sie auch die Zeilenumbrüche.

Führen Sie die Abfrage aus, und ermitteln Sie, welche Auswirkungen der mehrzeilige Kommentar auf das Ergebnis der Abfrage hat.

Lösung zu Übung 1

```
-- Alle Spalten der Tabelle mitarbeiter abfragen
SELECT * FROM mitarbeiter;
```

Listing 1.8 Verwendung eines einzeiligen Kommentars in SQL

Der einzeilige Kommentar hat keine Auswirkungen auf das Ergebnis der Abfrage. Die Datenbank liefert Ihnen wie gewohnt eine Ergebnistabelle gemäß Ihrer **SELECT**-Abfrage zurück.

Lösung zu Übung 2

```
/*
Programmierer: Ihr Name
Datum: Das aktuelle Datum
Funktion: Filtern aller Mitarbeiter, die zur Abteilung Einkauf gehören
*/
SELECT
  name,vorname,abteilung
FROM
  mitarbeiter
WHERE
  abteilung='Einkauf';
```

Listing 1.9 Verwendung eines mehrzeiligen Kommentars in SQL

Der mehrzeilige Kommentar hat bei der Ausführung der **SELECT**-Anweisung keinerlei Auswirkungen auf das Ergebnis der Abfrage.

Kapitel 10

Operationen auf Tabellen in Beziehungen anwenden

Ein relationales Datenbanksystem stellt sicher, dass die Beziehungen zwischen den Tabellen auch stets berücksichtigt werden. Wie wir uns dies zunutze machen, sehen Sie in diesem Kapitel.

In diesem Kapitel führen wir das Beispiel aus Kapitel 9 fort und beschäftigen uns weiter mit der *Ausbildungsdatenbank*.

10.1 Zeilen in Tabellen einfügen, die in Beziehung zueinander stehen

Um Abhängigkeiten zwischen Tabellen zu beschreiben, verwenden wir auch hier die vereinfachte Ausdrucksweise der Eltern-Kind-Beziehung. Tabellen, die sich über eine *Fremdschlüsseldefinition* in der **CREATE TABLE**-Anweisung in Abhängigkeit zu einer anderen Tabelle befinden, bezeichnen wir als *Kindtabellen*. Tabellen, die sich in keiner Abhängigkeit befinden, bezeichnen wir als *Elterntabellen*. In diesem Fall wären also die Tabellen *auszubildender* und *ausbildungsberuf* Beispiele für Elterntabellen, während *adresse* eine Kindtabelle darstellt, die in Beziehung zu einer anderen Tabelle steht.

10.1.1 Zeilen in die Tabelle »auszubildender« einfügen

Als Erstes fügen wir Zeilen in die Tabelle *auszubildender* ein. Die Tabelle *auszubildender* hat keinerlei Abhängigkeiten (**FOREIGN KEY CONSTRAINTS**), die wir beachten müssen, wenn wir neue Zeilen einfügen. Es ist lediglich darauf zu achten, dass für jeden Auszubildenden ein eindeutiger Primärschlüsselwert in der **INSERT**-Anweisung notiert wird. Listing 10.1 enthält sechs **INSERT**-Anweisungen, mit denen wir sechs neue Auszubildende in die Tabelle *auszubildender* eintragen:

```
INSERT INTO auszubildender (ausid,name,vorname,geburtsdatum)
  VALUES (1,'Müller','Ralf','2001.04.01');
INSERT INTO auszubildender (ausid,name,vorname,geburtsdatum)
  VALUES (2,'Klein','Sabine','2002.05.10');
```

```
INSERT INTO auszubildender (ausid,name,vorname,geburtsdatum)
  VALUES (3,'Lang','Peter','2001.03.11');
INSERT INTO auszubildender (ausid,name,vorname,geburtsdatum)
  VALUES (4,'Berg','Frank','2002.07.20');
INSERT INTO auszubildender (ausid,name,vorname,geburtsdatum)
  VALUES (5,'Erde','Sabine','2001.01.23');
INSERT INTO auszubildender (ausid,name,vorname,geburtsdatum)
  VALUES (6,'Grün','Justus','2001.04.15');
```

Listing 10.1 Zeilen in die Tabelle »auszubildender« einfügen

10.1.2 Zeilen in die Tabelle »ausbildungsberuf« einfügen

Als Nächstes fügen wir Zeilen in die Tabelle *ausbildungsberuf* ein. Die Tabelle *ausbildungsberuf* verfügt ebenfalls über keinerlei Abhängigkeiten (**FOREIGN CONSTRAINTS**), die wir beachten müssen. Auch hier ist lediglich darauf zu achten, dass für jeden Ausbildungsberuf ein eindeutiger Primärschlüsselwert in der **INSERT**-Anweisung eingetragen wird. Listing 10.2 enthält sechs **INSERT**-Anweisungen, mit denen wir sechs neue Ausbildungsberufe in die Tabelle *ausbildungsberuf* eintragen:

```
INSERT INTO ausbildungsberuf (berufsid,berufsbezeichnung)
  VALUES(1,'Energieelektroniker');
INSERT INTO ausbildungsberuf (berufsid,berufsbezeichnung)
  VALUES(2,'Mechatroniker');
INSERT INTO ausbildungsberuf (berufsid,berufsbezeichnung)
  VALUES(3,'Buchhalter');
INSERT INTO ausbildungsberuf (berufsid,berufsbezeichnung)
  VALUES(4,'Industriekaufmann');
INSERT INTO ausbildungsberuf (berufsid,berufsbezeichnung)
  VALUES(5,'Schlosser');
INSERT INTO ausbildungsberuf (berufsid,berufsbezeichnung)
  VALUES(6,'Elektriker');
```

Listing 10.2 Zeilen in die Tabelle »ausbildungsberuf« einfügen

10.1.3 Zeilen in die Tabelle »lehrfach« einfügen

Bei der Tabelle *lehrfach* handelt es sich ebenfalls um eine nicht abhängige Tabelle. Wir können also neue Zeilen in die Tabelle *lehrfach* einfügen, ohne auf irgendwelche Abhängigkeiten (**FOREIGN KEY CONSTRAINTS**) achten zu müssen. Es ist nur wichtig, dass eindeutige Primärschlüsselwerte eingetragen werden. Listing 10.3 enthält zehn **INSERT**-Anweisungen, mit denen wir zehn neue Lehrfächer in die Tabelle *lehrfach* eintragen:


```
INSERT INTO lehrfach (lehrfachid,lehrfach)
  VALUES (1,'Mathematik');
INSERT INTO lehrfach (lehrfachid,lehrfach)
  VALUES (2,'Buchhaltung 1');
INSERT INTO lehrfach (lehrfachid,lehrfach)
  VALUES (3,'Buchhaltung 2');
INSERT INTO lehrfach (lehrfachid,lehrfach)
  VALUES (4,'Mechanik Grundlagen 1');
INSERT INTO lehrfach (lehrfachid,lehrfach)
  VALUES (5,'Mechanik Grundlagen 2');
INSERT INTO lehrfach (lehrfachid,lehrfach)
  VALUES (6,'Englisch');
INSERT INTO lehrfach (lehrfachid,lehrfach)
  VALUES (7,'Elektronik Grundlagen 1');
INSERT INTO lehrfach (lehrfachid,lehrfach)
  VALUES (8,'Elektronik Grundlagen 2');
INSERT INTO lehrfach (lehrfachid,lehrfach)
  VALUES (9,'Rechnungsbearbeitung 1');
INSERT INTO lehrfach (lehrfachid,lehrfach)
  VALUES (10,'Rechnungsbearbeitung 2');
```

Listing 10.3 Zeilen in die Tabelle »lehrfach« einfügen

10.1.4 Zeilen in die Tabelle »adresse« (inklusive der Beziehungen) einfügen

Die Tabelle *adresse* befindet sich in einer 1:n-Abhängigkeit zu der Tabelle *auszubildender*. Neue Zeilen, die wir hier mit einer **INSERT**-Anweisung eintragen, müssen also auf gültige Primärschlüsselwerte in der Tabelle *auszubildender* verweisen. Die sechs **INSERT**-Anweisungen in Listing 10.4 sorgen dafür, dass sechs Adressen in die Tabelle *adresse* eingetragen werden. Die ersten drei **INSERT**-Anweisungen werden mit den Fremdschlüsselwerten 1, 3 und 5 ausgestattet und verweisen somit in der Tabelle *auszubildender* auf die Zeilen mit den Primärschlüsselwerten 1, 3 und 5. Die letzten drei **INSERT**-Anweisungen enthalten keine Fremdschlüsselwerte. Hier wurde anstelle eines Fremdschlüsselwerts der nicht definierte Wert **NULL** eingetragen. Die letzten drei Zeilen verweisen also auf keine Auszubildenden. Es handelt sich stattdessen um Adressen, die nicht zugeordnet sind.

```
/* Mit Zuordnung zu Auszubildenden */
INSERT INTO adresse (aid,strasse,nr,plz,ort,fk_ausid)
  VALUES (1,'Mondstraße','8',50827,'Köln',1);
INSERT INTO adresse (aid,strasse,nr,plz,ort,fk_ausid)
  VALUES (2,'Sternstraße','10',50127,'Bonn',3);
INSERT INTO adresse (aid,strasse,nr,plz,ort,fk_ausid)
  VALUES (3,'Sonnenstraße','1',50129,'Bonn',5);
```

```
/*Ohne Zuordnung zu Auszubildenden*/
INSERT INTO adresse (aid,strasse,nr,plz,ort,fk_ausid)
  VALUES (4,'Jupiterstraße','11',50827,'Köln',NULL);
INSERT INTO adresse (aid,strasse,nr,plz,ort,fk_ausid)
  VALUES (5,'Uranusstraße','9',50127,'Bonn',NULL);
INSERT INTO adresse (aid,strasse,nr,plz,ort,fk_ausid)
  VALUES (6,'Marsstraße','9',50129,'Bonn',NULL);
```

Listing 10.4 Neue Zeilen in die Tabelle »adresse« einfügen

10.1.5 Zeilen in die Tabelle »ausbildungsvertrag« (inklusive der Beziehungen) einfügen

Die Tabelle *ausbildungsvertrag* steht jeweils in einer 1:1-Beziehung zu den Tabellen *auszubildender* und *ausbildungsberuf*. Neue Zeilen, die wir mit einer **INSERT**-Anweisung eintragen, müssen also auf gültige Primärschlüsselwerte der Tabellen *auszubildender* und *ausbildungsberuf* verweisen. Listing 10.5 enthält sechs **INSERT**-Anweisungen. Die letzten beiden Werte, die in der **VALUES**-Klausel enthalten sind, verweisen mit ihren Fremdschlüsselwerten der Spalten *fk_ausid* und *fk_berufsid* auf die Primärschlüsselwerte der Tabellen *auszubildender* (Primärschlüsselspalte: *ausid*) und *ausbildungsberuf* (Primärschlüsselspalte: *berufsid*). Jede Zeile, die wir hier einfügen, enthält also eine Beziehung zu den Zeilen in den Tabellen *auszubildender* und *ausbildungsberuf*.

```
INSERT INTO ausbildungsvertrag (vid,vertragsdatum,fk_ausid,fk_berufsid)
  VALUES (1,'2015.06.01',5,2);
INSERT INTO ausbildungsvertrag (vid,vertragsdatum,fk_ausid,fk_berufsid)
  VALUES (2,'2015.06.01',4,4);
INSERT INTO ausbildungsvertrag (vid,vertragsdatum,fk_ausid,fk_berufsid)
  VALUES (3,'2015.06.01',1,3);
INSERT INTO ausbildungsvertrag (vid,vertragsdatum,fk_ausid,fk_berufsid)
  VALUES (4,'2015.06.01',3,1);
INSERT INTO ausbildungsvertrag (vid,vertragsdatum,fk_ausid,fk_berufsid)
  VALUES (5,'2015.06.01',6,1);
INSERT INTO ausbildungsvertrag (vid,vertragsdatum,fk_ausid,fk_berufsid)
  VALUES (6,'2015.06.01',2,5);
```

Listing 10.5 Zeilen in die Tabelle »ausbildungsvertrag« einfügen

10.1.6 Zeilen in die Tabelle »beruflehrfach« (inklusive der Beziehungen) einfügen

Die Tabelle *beruflehrfach* bildet eine m:n-Beziehung zwischen den Tabellen *ausbildungsberuf* und *lehrfach* ab. In der Tabelle *beruflehrfach* werden Kombinationen von

Fremdschlüsselwerten gespeichert, die jeweils auf die Primärschlüsselwerte der Tabellen *ausbildungsberuf* und *lehrfach* verweisen.

Die Fremdschlüsselspalte *fk_berufsid* der Tabelle *beruflehrfach* verweist auf die Primärschlüsselspalte *berufsid* der Tabelle *ausbildungsberuf*. Die Fremdschlüsselspalte *fk_lehrfachid* wiederum verweist auf die Primärschlüsselspalte *lehrfachid* der Tabelle *lehrfach*. In Listing 10.6 sehen Sie zwölf **INSERT**-Anweisungen, mit denen Kombinationen von Fremdschlüsselwerten gespeichert werden, um entsprechende Beziehungen zu den Zeilen der Tabellen *ausbildungsberuf* und *lehrfach* herzustellen:

```
INSERT INTO beruflehrfach (fk_berufsid,fk_lehrfachid)
  VALUES (1,1);
INSERT INTO beruflehrfach (fk_berufsid,fk_lehrfachid)
  VALUES (1,6);
INSERT INTO beruflehrfach (fk_berufsid,fk_lehrfachid)
  VALUES (1,7);
INSERT INTO beruflehrfach (fk_berufsid,fk_lehrfachid)
  VALUES (1,8);
INSERT INTO beruflehrfach (fk_berufsid,fk_lehrfachid)
  VALUES (4,2);
INSERT INTO beruflehrfach (fk_berufsid,fk_lehrfachid)
  VALUES (4,3);
INSERT INTO beruflehrfach (fk_berufsid,fk_lehrfachid)
  VALUES (4,6);
INSERT INTO beruflehrfach (fk_berufsid,fk_lehrfachid)
  VALUES (4,9);
INSERT INTO beruflehrfach (fk_berufsid,fk_lehrfachid)
  VALUES (6,1);
INSERT INTO beruflehrfach (fk_berufsid,fk_lehrfachid)
  VALUES (6,4);
INSERT INTO beruflehrfach (fk_berufsid,fk_lehrfachid)
  VALUES (6,7);
INSERT INTO beruflehrfach (fk_berufsid,fk_lehrfachid)
  VALUES (6,8);
```

Listing 10.6 Zeilen in die Tabelle »beruflehrfach« einfügen

10.1.7 Zeilen in die Tabelle »mitarbeiterausbildungsbetrieb« (inklusive der Beziehungen) einfügen

Die Tabelle *mitarbeiterausbildungsbetrieb* bildet eine 1:n-Beziehung mit sich selbst. Die Tabelle enthält eine Fremdschlüsselspalte *fk_mitarbeiterid*, die auf die Primärschlüsselspalte *mitarbeiterid* referenziert.

Listing 10.7 enthält zehn **INSERT**-Anweisungen, in denen jeweils Werte der spalte *fk_mitarbeiterid* auf Werte der Spalte *mitarbeiterid* verweisen:

```
INSERT INTO mitarbeiterausbildungsbetrieb (mitarbeiterid,name,vorname,
fk_mitarbeiterid)
  VALUES (1,'Müller','Alfred',NULL);
INSERT INTO mitarbeiterausbildungsbetrieb (mitarbeiterid,name,vorname,
fk_mitarbeiterid)
  VALUES (2,'Ungern','Peter',1);
INSERT INTO mitarbeiterausbildungsbetrieb (mitarbeiterid,name,vorname,
fk_mitarbeiterid)
  VALUES (3,'Erdenschein','Claudia',1);
INSERT INTO mitarbeiterausbildungsbetrieb (mitarbeiterid,name,vorname,
fk_mitarbeiterid)
  VALUES (4,'Sternenschein','Ute',1);
INSERT INTO mitarbeiterausbildungsbetrieb (mitarbeiterid,name,vorname,
fk_mitarbeiterid)
  VALUES (5,'Augustus','Frank',1);
INSERT INTO mitarbeiterausbildungsbetrieb (mitarbeiterid,name,vorname,
fk_mitarbeiterid)
  VALUES (6,'Erdenfels','Christine',NULL);
INSERT INTO mitarbeiterausbildungsbetrieb (mitarbeiterid,name,vorname,
fk_mitarbeiterid)
  VALUES (7,'Hoffnung','Ralf',6);
INSERT INTO mitarbeiterausbildungsbetrieb (mitarbeiterid,name,vorname,
fk_mitarbeiterid)
  VALUES (8,'Freud','Erika',6);
INSERT INTO mitarbeiterausbildungsbetrieb (mitarbeiterid,name,vorname,
fk_mitarbeiterid)
  VALUES (9,'Bergfels','Diether',6);
INSERT INTO mitarbeiterausbildungsbetrieb (mitarbeiterid,name,vorname,
fk_mitarbeiterid)
  VALUES (10,'Lemon','Reinhold',6);
```

Listing 10.7 Zeilen in die Tabelle »mitarbeiterausbildungsbetrieb« einfügen

[*]

Zusammenfassung: Zeilen, die in Beziehung stehen, in Tabellen einfügen

Wenn Zeilen in *Kindtabellen*, die in Beziehung zu Zeilen aus *Elterntabellen* stehen, eingefügt werden sollen, gilt es, eine Reihenfolge einzuhalten.

Zuerst werden die Zeilen in eine Elterntabelle eingefügt. Dann werden die Zeilen mit den entsprechenden *Fremdschlüsselwerten* in Kindtabellen eingefügt.

10.1.8 Übungen

Übung 1

In Kapitel 9, »Datenmodelle in Tabellen überführen«, haben Sie Tabellen erstellt, die Informationen für eine Bibliothek verwalten sollen. In dieser Übung sollen für die Tabellen *fachbereich* und *verlag*, die starke Entitäten darstellen, **INSERT**-Anweisungen formuliert werden, mit denen Sie Zeilen in die Tabellen einfügen. Die Tabellen sind also autark. Sie müssen hier keine Beziehungen zu Zeilen in anderen Tabellen beachten. In Tabelle 10.1 sehen Sie Zeilen, die für die Tabelle *verlag* vorgesehen sind. Tabelle 10.2 enthält wiederum Zeilen, die für die Tabelle *fachbereich* bestimmt sind.

verlagid	verlag
1	Rheinwerk Verlag
2	Elektro Verlag
3	Mechanik Verlag
4	Kaufmann Verlag
5	Medien Verlag

Tabelle 10.1 Zeilen für die Tabelle »verlag«

fachbereichid	fachbereich
1	Elektrotechnik
2	Kaufmann
3	Mechanik
4	Pneumatik

Tabelle 10.2 Zeilen für die Tabelle »fachbereich«

Übung 2

Fügen Sie Zeilen in die Tabelle *fachbuch* ein, die mit einem Fremdschlüsselwert auf Zeilen in der Tabelle *verlag* referenzieren. In Übung 1 haben Sie fünf Verlage in die Tabelle *verlag* eingefügt. Uns stehen also fünf Primärschlüsselwerte aus der Tabelle *verlag* zur Verfügung, die wir als Fremdschlüsselwerte in der Tabelle *fachbuch* verwenden können, je nachdem, von welchem Verlag ein Fachbuch verlegt wird. Beachten Sie, dass die Fremdschlüsselwerte in die Spalte *fk_verlagid* eingefügt werden.

Diese Fremdschlüsselwerte referenzieren auf Zeilen, die Sie in Tabelle 10.3 für **INSERT**-Anweisungen für die Tabelle *verlag* genutzt haben. Die letzten drei Zeilen mit den Primärschlüsselwerten 7, 8 und 9 der Spalte *fachbuchid* verfügen über keine Referenz, die über Fremdschlüsselwerte festgelegt wurde. Tragen Sie hier jeweils **NULL** in die **VALUES**-Klausel der **INSERT**-Anweisung ein.

fachbuchid	isbn	titel	fk_verlagid
1	1235	Mechanik	3
2	9878	Elektrotechnik	2
3	2323	Elektronik	2
4	2254	Pneumatik	3
5	4455	Mathematik Grundlagen 1	2
6	4456	Mathematik Grundlagen 2	2
7	5566	Mengenlehre	**NULL**
8	7766	Kommunikation 1	**NULL**
9	7767	Kommunikation 2	**NULL**

Tabelle 10.3 Zeilen für die Tabelle »fachbuch«

Übung 3

Jetzt fehlt uns noch die Zuordnung eines Fachbuchs zu einem Fachbereich. Diese Zuordnung können Sie mit der Schlüsseltabelle *fachbereichfachbuch* realisieren. Hier müssen jeweils gültige Primärschlüsselwerte oder Kombinationen aus den Tabellen *fachbuch* und *fachbereich* eingetragen werden, um die Beziehung eines Schlüsselpaars zu den Zeilen der Tabellen *fachbuch* bzw. *fachbereich* herzustellen.

In Tabelle 10.4 sehen Sie die Zuordnungen über die Fremdschlüsselwerte eines Fachbuchs zu einem Fachbereich. Formulieren Sie **INSERT**-Anweisungen, mit denen Sie die jeweiligen Zuordnungen in die Tabelle *fachbereichfachbuch* eintragen.

fk_fachbereichid	fk_fachbuchid
3	1
1	3
4	4

Tabelle 10.4 Zeilen für die Tabelle »fachbereichfachbuch«

fk_fachbereichid	fk_fachbuchid
1	5
1	6
1	7
2	8
2	9

Tabelle 10.4 Zeilen für die Tabelle »fachbereichfachbuch« (Forts.)

Lösung zu Übung 1

```
/* Zeilen in die Tabelle verlag einfügen */
INSERT INTO verlag (verlagid,verlag)
VALUES (1, 'Rheinwerk Verlag');
INSERT INTO verlag (verlagid,verlag)
VALUES (2, 'Elektro Verlag');
INSERT INTO verlag (verlagid,verlag)
VALUES (3, 'Mechanik Verlag');
INSERT INTO verlag (verlagid,verlag)
VALUES (4, 'Kaufmann Verlag');
INSERT INTO verlag (verlagid,verlag)
VALUES (5, 'Medien Verlag');

/* Zeilen in die Tabelle fachbereich einfügen */
INSERT INTO fachbereich (fachbereichid,fachbereich)
VALUES (1, 'Elektrotechnik');
INSERT INTO fachbereich (fachbereichid,fachbereich)
VALUES (2, 'Kaufmann');
INSERT INTO fachbereich (fachbereichid,fachbereich)
VALUES (3, 'Mechanik');
INSERT INTO fachbereich (fachbereichid,fachbereich)
VALUES (4, 'Pneumatik');
```

Listing 10.8 INSERT-Anweisungen zum Einfügen von Zeilen in die Tabellen »verlag« und »fachbereich«

Lösung zu Übung 2

```
INSERT INTO fachbuch (fachbuchid,isbn,titel,fk_verlagid)
VALUES (1,'1235','Mechanik',3);
```

```
INSERT INTO fachbuch (fachbuchid,isbn,titel,fk_verlagid)
VALUES (2,'9878','Elektrotechnik',2);
INSERT INTO fachbuch (fachbuchid,isbn,titel,fk_verlagid)
VALUES (3,'2323','Elektronik',2);
INSERT INTO fachbuch (fachbuchid,isbn,titel,fk_verlagid)
VALUES (4,'2254','Pneumatik',3);
INSERT INTO fachbuch (fachbuchid,isbn,titel,fk_verlagid)
VALUES (5,'4455','Mathematik Grundlagen 1',2);
INSERT INTO fachbuch (fachbuchid,isbn,titel,fk_verlagid)
VALUES (6,'4456','Mathematik Grundlagen 2',2);
INSERT INTO fachbuch (fachbuchid,isbn,titel,fk_verlagid)
VALUES (7,'5566','Mengenlehre',NULL);
INSERT INTO fachbuch (fachbuchid,isbn,titel,fk_verlagid)
VALUES (8,'7766','Kommunikation 1',NULL);
INSERT INTO fachbuch (fachbuchid,isbn,titel,fk_verlagid)
VALUES (9,'7767','Kommunikation 2',NULL);
```

Listing 10.9 INSERT-Anweisungen zum Einfügen von Zeilen in die Tabelle »fachbuch«

Lösung zu Übung 3

```
INSERT INTO fachbereichfachbuch (fk_fachbereichid,fk_fachbuchid)
VALUES (3,1);
INSERT INTO fachbereichfachbuch (fk_fachbereichid,fk_fachbuchid)
VALUES (1,3);
INSERT INTO fachbereichfachbuch (fk_fachbereichid,fk_fachbuchid)
VALUES (4,4);
INSERT INTO fachbereichfachbuch (fk_fachbereichid,fk_fachbuchid)
VALUES (1,5);
INSERT INTO fachbereichfachbuch (fk_fachbereichid,fk_fachbuchid)
VALUES (1,6);
INSERT INTO fachbereichfachbuch (fk_fachbereichid,fk_fachbuchid)
VALUES (1,7);
INSERT INTO fachbereichfachbuch (fk_fachbereichid,fk_fachbuchid)
VALUES (2,8);
INSERT INTO fachbereichfachbuch (fk_fachbereichid,fk_fachbuchid)
VALUES (2,9);
```

Listing 10.10 INSERT-Anweisungen zum Einfügen von Zeilen in die Tabelle »fachbereichfachbuch«

10.2 Zeilen aus Tabellen, die in Beziehung stehen, mit JOIN verbunden abfragen

Wir haben Tabellen in Beziehung zueinander gesetzt und Datensätze in Tabellen eingefügt, die berücksichtigen, dass sie in Beziehung zueinander stehen. Jetzt schauen wir uns an, wie Sie Datensätze aus solchen Tabellen abfragen. Daher nutzen wir die Beziehungen zwischen den Tabellen, um **SELECT**-Abfragen zu formulieren, mit denen wir die Datensätze, deren Schlüssel gleich sind, verbunden abfragen. Das heißt, dass Sie in der Spaltenauswahlliste die Wahl haben zwischen den Spalten, die in den verbundenen Tabellen vorkommen.

Es gibt vier unterschiedliche **JOIN**-Arten, mit denen Sie Tabellen abfragen können, deren Zeilen durch Fremdschlüsselbeziehungen verbunden sind: der **INNER JOIN**, der **LEFT OUTER JOIN**, der **RIGHT OUTER JOIN** und der **FULL OUTER JOIN**. Wenn Sie einen **INNER JOIN** verwenden, werden die Zeilen aus Tabellen verbunden abgefragt, wenn Schlüsselwerte übereinstimmen. Ein **LEFT OUTER JOIN** verbindet hingegen alle Zeilen der linken Tabelle mit allen Zeilen der rechten Tabelle. Außerdem bleiben Ihnen beim **LEFT OUTER JOIN** alle Zeilen erhalten, die in der linken Tabelle existent sind, aber in der rechten Tabelle nicht mit Schlüsselwerten verbunden werden können. Für den **RIGHT OUTER JOIN** gilt das Ganze umgekehrt. Ein **FULL OUTER JOIN** verbindet Zeilen mit übereinstimmenden Schlüsselwerten. In die Ergebnistabelle werden auch Zeilen der linken und rechten Tabelle aufgenommen, die keine übereinstimmenden Schlüsselwerte haben. Ein **CROSS JOIN** schließlich verbindet alle Zeilen der beiden Tabellen unabhängig aller Schlüsselbeziehungen miteinander.

Die Diagramme in Abbildung 10.1 bis Abbildung 10.5 veranschaulichen die Operationen:

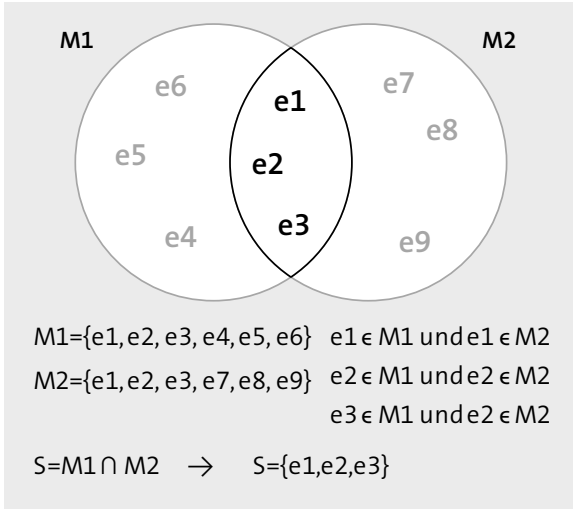
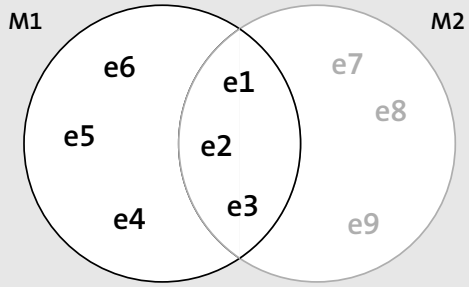


Abbildung 10.1 Ein INNER JOIN

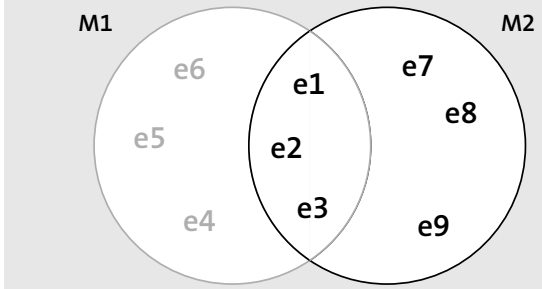


M1={e1, e2, e3, e4, e5, e6}

M2={e1, e2, e3, e7, e8, e9}

S=M1 ∩ M2 → S={e1, e2, e3}
R=M1 \ M2 → R={e6, e5, e4}
L=S ∪ R → R={e1, e2, e3, e4, e5, e6}

Abbildung 10.2 Ein LEFT OUTER JOIN

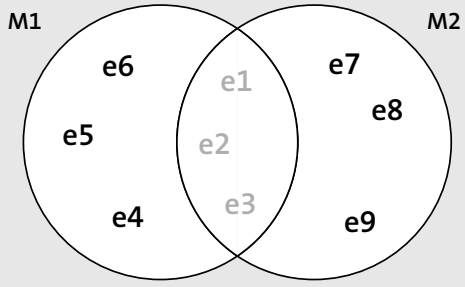


M1={e1, e2, e3, e4, e5, e6}

M2={e1, e2, e3, e7, e8, e9}

S=M1 ∩ M2 → S={e1, e2, e3}
R=M2 \ M1 → R={e7, e8, e9}
L=S ∪ R → R={e1, e2, e3, e7, e8, e9}

Abbildung 10.3 Ein RIGHT OUTER JOIN

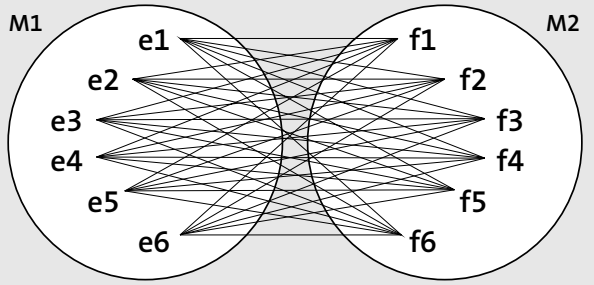


M1={e1, e2, e3, e4, e5, e6}

M2={e1, e2, e3, e7, e8, e9}

S=M1 ∩ M2 → S={e1, e2, e3}
R1=M1 \ M2 → R1={e4, e5, e6}
R2=M2 \ M1 → R2={e7, e8, e9}
F=S ∪ R1 ∪ R2 → F={e1, e2, e3, e4, e5, e6, e7, e8, e9 }

Abbildung 10.4 Ein FULL OUTER JOIN



M1={e1, e2, e3, e4, e5, e6}

M2={f1, f2, f3, f4, f5, f6}

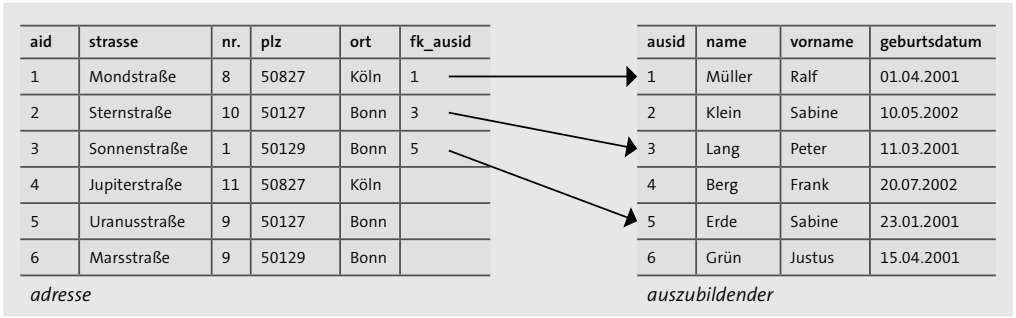
K=M1 X M2 → K={(e1, f1), (e1, f2), (e1, f3), ..., (e6, f6)}

Abbildung 10.5 Ein CROSS JOIN

10.2.1 Zeilen mit einem INNER JOIN verbinden

Ein `INNER JOIN` verbindet Zeilen von Tabellen, indem auf Gleichheit von Primärschlüssel- und Fremdschlüsselwerten geprüft wird. Wie bereits erläutert, stehen die Tabellen *auszubildender* und *adresse* über eine Fremdschlüsselbeziehung in der Tabelle *adresse* zueinander in Beziehung.

In Abbildung 10.6 sehen Sie, dass jeweils die Fremdschlüsselwerte der Spalte *fk_ausid* der Tabelle *adresse* auf gleiche Primärschlüsselwerte der Spalte *ausid* in der Tabelle *auszubildender* verweisen.



aid	strasse	nr.	plz	ort	fk_ausid
1	Mondstraße	8	50827	Köln	1
2	Sternstraße	10	50127	Bonn	3
3	Sonnenstraße	1	50129	Bonn	5
4	Jupiterstraße	11	50827	Köln	
5	Uranusstraße	9	50127	Bonn	
6	Marsstraße	9	50129	Bonn	

adresse

ausid	name	vorname	geburtsdatum
1	Müller	Ralf	01.04.2001
2	Klein	Sabine	10.05.2002
3	Lang	Peter	11.03.2001
4	Berg	Frank	20.07.2002
5	Erde	Sabine	23.01.2001
6	Grün	Justus	15.04.2001

auszubildender

Abbildung 10.6 Zeilen von Tabellen mit Schlüsselwerten verbinden

Diese Tabellen, die in Beziehung zueinander stehen, wollen wir mit einer **SELECT**-Anweisung abfragen. Wenn wir zwei Tabellen abfragen, stehen uns natürlich auch die Spalten aus zwei Tabellen zur Verfügung. Sie realisieren die Verbindung von Zeilen über identische Schlüsselwerte aus Tabellen mit einem **INNER JOIN**. Ein **INNER JOIN** verbindet Zeilen von Tabellen nur dann, wenn eine Schlüsselbeziehung zwischen einer Eltern- und einer Kindtabelle existiert. Wenn Zeilen in den Tabellen vorhanden sind, die über keine Beziehung verfügen, werden sie also nicht von einem **INNER JOIN** berücksichtigt.

In Listing 10.11 sehen Sie eine **SELECT**-Abfrage mit einem **INNER JOIN**:

```
SELECT name,vorname,plz,ort
FROM auszubildender INNER JOIN adresse
ON ausid=fk_ausid;
```

Listing 10.11 Die Tabelle »auszubildender« und »adresse« mit einem INNER JOIN verbunden abfragen

Hinter dem Schlüsselwort **SELECT** sehen Sie wie gewohnt eine Spaltenauswahlliste. Hier gibt es nur eine Neuerung: Die Spalten *name* und *vorname* fragen wir aus der Tabelle *auszubildender* ab. Die Spalten *plz* und *ort* stammen hingegen aus der Tabelle *adresse*. Es folgen das Schlüsselwort **FROM** und die Tabellenbezeichnung *auszubildender*. Bis hierhin haben Sie fast nichts Neues kennengelernt.

Hinter der Tabellenbezeichnung *auszubildender* folgen jetzt die neuen Schlüsselwörter **INNER JOIN** (innerer Verbund). Hinter der **INNER JOIN**-Klausel notieren Sie die Tabelle, die über einen **INNER JOIN** verbunden abgefragt werden soll. Somit haben Sie schon einmal bekannt gemacht, welche Tabellen verbunden abgefragt werden sollen.

Jetzt müssen wir nur noch angeben, wie die Tabellen miteinander verbunden werden sollen. Die Bedingung, die auf Gleichheit von Schlüsselwerten prüft (Primärschlüssel- und Fremdschlüsselwerte), legen Sie in einer **ON**-Klausel fest.

Das Schlüsselwort **INNER** der **INNER JOIN**-Klausel ist optional anzugeben. In Listing 10.12 sehen Sie eine **SELECT**-Abfrage mit einem **INNER JOIN**, in dem auf das Schlüsselwort **INNER** verzichtet wurde. Diese Abfrage liefert Ihnen exakt das gleiche Ergebnis wie die Abfrage aus Listing 10.11.

```
SELECT name,vorname,plz,ort
FROM auszubildender JOIN adresse
ON ausid=fk_ausid;
```

Listing 10.12 Einen INNER JOIN ohne das optionale Schlüsselwort INNER verwenden

Tabelle 10.5 zeigt Ihnen das Ergebnis der Abfrage:

name	vorname	plz	ort
Müller	Ralf	50827	Köln
Lang	Peter	50127	Bonn
Erde	Sabine	50129	Bonn

Tabelle 10.5 Ergebnistabelle für eine Abfrage mit einem INNER JOIN

Die Zeilen, die jeweils durch eine Referenz der Kindtabelle *adresse* auf Zeilen der Elterntabelle *auszubildender* verweisen, konnten erfolgreich verbunden werden. Sicher fällt Ihnen auf, dass die Datensätze, die über keine Referenz (Fremdschlüsselwert gleich **NULL**) aus der Tabelle *adresse* verfügen, nicht verbunden werden konnten. Wie Sie Zeilen, die über keine Schlüsselbeziehungen verfügen, mit in die Ergebnisliste aufnehmen, werden Sie in den nächsten Abschnitten erfahren.

10.2.2 Zeilen mit einem LEFT OUTER JOIN verbinden

Ein **LEFT OUTER JOIN** verknüpft ebenfalls Zeilen von Tabellen über eine Fremdschlüsselbeziehung. Zunächst einmal verbindet ein **LEFT OUTER JOIN** alle Zeilen von zwei Tabellen über eine Schlüsselbeziehung. Immer dann, wenn ein Fremdschlüssel aus einer Kindtabelle auf einen Primärschlüssel aus einer Elterntabelle referenziert, werden Zeilen entsprechend verbunden. Das kennen Sie bereits vom **INNER JOIN**.

Ein **LEFT OUTER JOIN** gibt zusätzlich die Zeilen der Tabelle *auszubildender* (die links von der **LEFT OUTER JOIN**-Klausel notiert ist) aus, die nicht über Schlüsselwerte mit der Tabelle *adresse* (die rechts von der **LEFT OUTER JOIN**-Klausel notiert ist) verbunden werden können. Diese Zeilen bleiben uns also erhalten. In diesem Fall bedeutet das, dass

die Spaltenwerte der Zeilen der Elterntabelle *auszubildender* ausgegeben werden, auf die keine Referenzierung aus den Zeilen der Kindtabelle *adresse* erfolgt.

LEFT OUTER JOIN und Schlüsselbeziehungen

Beachten Sie Folgendes: Einem **LEFT OUTER JOIN** ist es grundsätzlich egal, ob Sie Fremdschlüsselwerte mit Primärschlüsselwerten vergleichen. In der **ON**-Klausel wird eine Bedingung ausgewertet. Wenn diese wahr ist, werden Zeilen verbunden ausgegeben. Wenn sie nicht wahr ist, wird keine Zeile verbunden abgefragt. Das war es auch schon.

Der **LEFT OUTER JOIN** fragt also auch die Zeilen aus der Tabelle links von der **LEFT OUTER JOIN**-Klausel ab, die nicht über eine passende Schlüsselbeziehung mit der rechten Tabelle verbunden werden können. Die Zeilen der Tabelle links von der **LEFT OUTER JOIN**-Klausel sind daher ebenso in der Ergebnistabelle enthalten.

In Abbildung 10.6 sehen Sie unsere bekannte Tabelle *auszubildender*. Darin finden Sie jetzt insgesamt sechs Zeilen. Aus der Tabelle *adresse* wird nicht auf die Zeilen mit den Primärschlüsselwerten 2, 4 und 6 in der Spalte *ausid* referenziert. Diesen drei Auszubildenden kann derzeit noch kein Wohnort zugeordnet werden. In der Tabelle *adresse* sehen Sie also auch keine Zeilen mit Fremdschlüsselwerten, die auf die Zeilen der Tabelle *auszubildende*r mit Primärschlüsselwerten verweisen.

Die **SELECT**-Abfrage in Listing 10.13 verbindet Zeilen über einen **LEFT OUTER JOIN**:

```
SELECT name,vorname,plz,ort
FROM auszubildender LEFT OUTER JOIN adresse
ON ausid=fk_ausid;
```

Listing 10.13 Die Tabellen »auszubildender« und »adresse« mit einem LEFT OUTER JOIN verbunden abfragen

Die Zeilen aus der linken Elterntabelle *auszubildender*, die nicht mit der Kindtabelle *adresse* verbunden werden können, werden ebenso abgefragt. Anstelle des **INNER JOIN**-Schlüsselworts verwenden Sie jetzt das Schlüsselwort **LEFT OUTER JOIN**, um alle Zeilen der linken Tabelle abzufragen, die über Schlüsselwerte mit der rechten Tabelle verbunden oder eben nicht verbunden werden können.

Das Schlüsselwort **OUTER** der **LEFT OUTER JOIN**-Klausel ist dabei optional anzugeben. In Listing 10.14 sehen Sie eine **SELECT**-Abfrage mit einem **LEFT OUTER JOIN**, in dem auf das Schlüsselwort **OUTER** verzichtet wurde. Diese Abfrage liefert Ihnen exakt das gleiche Ergebnis wie die Abfrage aus Listing 10.13.

```
SELECT name,vorname,plz,ort
FROM auszubildender LEFT JOIN adresse
ON ausid=fk_ausid;
```

Listing 10.14 Einen LEFT OUTER JOIN ohne das optionale Schlüsselwort OUTER verwenden

Tabelle 10.6 zeigt Ihnen das Ergebnis der Abfrage aus Listing 10.14:

name	vorname	plz	ort
Müller	Ralf	50827	Köln
Klein	Sabine	NULL	NULL
Lang	Peter	50127	Bonn
Berg	Frank	NULL	NULL
Erde	Sabine	50129	Bonn
Grün	Justus	NULL	NULL

Tabelle 10.6 Ergebnistabelle für eine Abfrage mit einem LEFT OUTER JOIN

Sie sehen, dass die Zeilen der Tabelle *auszubildender*, denen über die Fremdschlüsselbeziehungen der Tabelle *adresse* kein Wohnort zugeordnet werden konnte, ebenfalls angezeigt werden. Für diese Zeilen existieren natürlich keine Angaben bzw. Spaltenwerte zum Wohnort.

Wenn ein **LEFT OUTER JOIN** verwendet wird und Zeilen in der linken Tabelle existieren, die nicht über eine Schlüsselbeziehung mit Zeilen der rechten Tabelle verbunden werden können, füllt die Datenbank die Spaltenwerte der rechten Tabelle mit **NULL**-Werten auf. Wie können wir nun die Auszubildenden ermitteln, denen keine Adresse zugeordnet ist? Sie haben eben erfahren, dass die Datenbank unter Verwendung eines **LEFT OUTER JOIN**s alle Spaltenwerte von Zeilen der rechten Tabelle, die nicht mit Zeilen aus der linken Tabelle verbunden werden können, mit **NULL**-Werten auffüllt. Mit einer einfachen **WHERE**-Klausel, die auf **NULL**-Werte prüft, können Sie jetzt die Auszubildenden ermitteln, denen keine Adresse zugeordnet ist. Die **WHERE**-Klausel notieren Sie wie in Listing 10.15 angegeben hinter der **ON**-Klausel, um die Spalte *plz* der Tabelle *adresse* auf **NULL** zu prüfen. Welche Spalte der Tabelle *adresse* Sie auf **NULL**-Werte prüfen, ist nicht bedeutend, da ja sämtliche Spaltenwerte der Tabelle *wohnort* mit **NULL**-Werten in der Ergebnistabelle aufgefüllt werden, wenn keine Referenzierung zu einem Auszubildenden bzw. einer Zeile der Tabelle *auszubildender* möglich ist.

```
SELECT name,vorname,plz,ort
FROM auszubildender LEFT OUTER JOIN adresse
ON ausid=fk_ausid
WHERE plz IS NULL;
```

Listing 10.15 Die Tabellen »auszubildender« und »adresse« mit einem LEFT OUTER JOIN verbunden abfragen und nur die Zeilen filtern, denen keine Adresse zugeordnet werden konnte

Tabelle 10.7 zeigt Ihnen das Ergebnis der Abfrage aus Listing 10.15. Gemäß der Bedingung der **WHERE**-Klausel sind jetzt nur noch diejenigen Zeilen der Tabelle *auszubildender* in der Ergebnistabelle enthalten, die nicht über Schlüsselwerte mit einer Adresse verbunden werden konnten.

name	vorname	plz	ort
Klein	Sabine	NULL	NULL
Berg	Frank	NULL	NULL
Grün	Justus	NULL	NULL

Tabelle 10.7 Ergebnistabelle für eine Abfrage mit einem LEFT OUTER JOIN und einer Bedingung, die eine Spalte der rechten Tabelle auf NULL-Werte prüft

10.2.3 Zeilen mit einem RIGHT OUTER JOIN verbinden

Ein **RIGHT OUTER JOIN** verbindet zunächst einmal wie ein **INNER JOIN** oder ein **LEFT OUTER JOIN** alle Zeilen von zwei Tabellen, die über Schlüsselwerte verbunden abgefragt werden können.

Wie sieht es nun aus, wenn wir eine Kindtabelle abfragen wollen, die Zeilen ohne Referenzierung auf die Elterntabelle enthält? In Abschnitt 10.1.4 haben Sie bereits Zeilen in die Tabelle *adresse* eingefügt, die nicht auf Fremdschlüsselwerte, sondern auf Primärschlüsselwerte aus Zeilen der Tabelle *auszubildender* verweisen. Im Beispiel enthalten diese Zeilen **NULL**-Werte. Diese Zeilen der Tabelle *adresse* stehen also schlicht und ergreifend nicht in einer Beziehung zu Zeilen der Tabelle *auszubildender*.

Ein **RIGHT OUTER JOIN** gibt auch die Zeilen der Tabelle *adresse* (die rechts von der **RIGHT OUTER JOIN**-Klausel notiert ist) aus, die nicht über Schlüsselwerte mit der Tabelle *auszubildender* (die links von der **RIGHT OUTER JOIN**-Klausel notiert ist) verbunden werden können. Diese Zeilen bleiben uns also erhalten.

In unserem Fall bedeutet das, dass die Spaltenwerte derjenigen Zeilen der Tabelle *adresse* ausgegeben werden, die über keine Referenzierung zu den Zeilen der Tabelle *auszubildender* verfügen.

RIGHT OUTER JOIN und Schlüsselbeziehungen

Beachten Sie auch bei dieser Beziehungsart Folgendes: Einem **RIGHT OUTER JOIN** ist es ebenfalls grundsätzlich egal, ob Sie Fremdschlüsselwerte mit Primärschlüsselwerten vergleichen. In der **ON**-Klausel wird eine Bedingung ausgewertet; wenn sie wahr ist, werden Zeilen verbunden ausgegeben. Wenn sie nicht wahr ist, wird keine Zeile verbunden zurückgegeben.

Der **RIGHT OUTER JOIN** fragt also auch die Zeilen aus der Tabelle rechts von der **RIGHT OUTER JOIN**-Klausel ab, die nicht über eine passende Schlüsselbeziehung verbunden werden können. Die Zeilen der Tabelle rechts von der **RIGHT OUTER JOIN**-Klausel sind daher ebenso in der Ergebnistabelle enthalten.

Betrachten Sie auch hier Abbildung 10.6. In der Tabelle *adresse* sind Zeilen mit den Primärschlüsselwerten 4, 5 und 6 der Spalte *aid* enthalten, für die keine Fremdschlüsselwerte eingefügt wurden. Hier gibt es also keine Referenzierung auf Primärschlüsselwerte der Tabelle *auszubildender* und somit keine gültigen Beziehungen.

Für den **RIGHT OUTER JOIN** gilt ebenfalls, dass das Schlüsselwort **OUTER** optional anzugeben ist. In Listing 10.17 sehen Sie eine **SELECT**-Abfrage mit einem **RIGHT OUTER JOIN**, in dem auf das Schlüsselwort **OUTER** verzichtet wurde. Diese Abfrage liefert Ihnen exakt das gleiche Ergebnis wie die Abfrage aus Listing 10.16.

```
SELECT name,vorname,plz,ort
FROM auszubildender RIGHT OUTER JOIN adresse
ON ausid=fk_ausid;
```

Listing 10.16 Die Tabellen »auszubildender« und »adresse« mit einem RIGHT OUTER JOIN verbunden abfragen

```
SELECT name,vorname,plz,ort
FROM auszubildender RIGHT JOIN adresse
ON ausid=fk_ausid;
```

Listing 10.17 Einen RIGHT OUTER JOIN ohne das optionale Schlüsselwort OUTER verwenden

Zunächst einmal werden auch hier die Zeilen mit übereinstimmenden Schlüsselwerten verbunden abgefragt. Außerdem werden Zeilen der rechten Tabelle ausgegeben, die keine Einträge (nicht definierte Werte, also **NULL**-Werte) in der Fremdschlüsselspalte enthalten. Die Spalten der linken Tabelle *auszubildender* werden hier wieder mit **NULL**-Werten aufgefüllt.

Das Ergebnis der Abfrage sehen Sie in Tabelle 10.8:

name	vorname	plz	ort
Müller	Ralf	50827	Köln
Lang	Peter	50127	Bonn
Erde	Sabine	50129	Bonn

Tabelle 10.8 Ergebnis für eine Abfrage mit einem RIGHT OUTER JOIN

name	vorname	plz	ort
NULL	NULL	50827	Köln
NULL	NULL	50127	Bonn
NULL	NULL	50129	Bonn

Tabelle 10.8 Ergebnis für eine Abfrage mit einem RIGHT OUTER JOIN (Forts.)

Die Zeilen der Tabelle *adresse*, die keine Fremdschlüsselwerte enthalten, werden ebenfalls angezeigt. Wenn ein **RIGHT OUTER JOIN** verwendet wird und die Zeilen der rechten Tabelle nicht über eine Schlüsselbeziehung mit Zeilen der linken Tabelle verbunden werden können, füllt die Datenbank die Spaltenwerte der linken Tabelle wie beim **LEFT OUTER JOIN** mit **NULL**-Werten auf.

Vielleicht fragen Sie sich jetzt auch, wie Sie ausschließlich die Adressen ermitteln können, die keinem Auszubildenden zugeordnet sind. Wie beim **LEFT OUTER JOIN** prüfen wir beim **RIGHT OUTER JOIN** einfach wieder auf Spaltenwerte aus der linken Tabelle *auszubildender*, deren Zeilen mit keinem Wohnort verbunden werden konnten und somit mit **NULL**-Werten gefüllt wurden. In der **WHERE**-Klausel in Listing 10.18 werden die Spaltenwerte der Spalte *name* auf **NULL** geprüft, um nur die Zeilen in der Ergebnistabelle anzuzeigen, die nicht über eine Schlüsselreferenzierung mit Zeilen von Auszubildenden verbunden werden konnten. Sie können hier auch wieder jede beliebige Spalte der Tabelle *auszubildender* auf **NULL**-Werte prüfen.

```
SELECT name,vorname,plz,ort
FROM auszubildender RIGHT OUTER JOIN adresse
ON ausid=fk_ausid
WHERE name IS NULL;
```

Listing 10.18 Die Tabellen »auszubildender« und »adresse« mit einem RIGHT OUTER JOIN verbunden abfragen und nur die Zeilen ausgeben, die in der rechten Tabelle über keine Schlüsselwerte verfügen

Tabelle 10.9 zeigt Ihnen das Ergebnis der Abfrage aus Listing 10.18.

name	vorname	plz	ort
NULL	NULL	50827	Köln
NULL	NULL	50127	Bonn
NULL	NULL	50129	Bonn

Tabelle 10.9 Ergebnistabelle für eine Abfrage mit einem RIGHT OUTER JOIN und einer Bedingung, die eine Spalte der rechten Tabelle auf NULL-Werte prüft

Gemäß der Bedingung der **WHERE**-Klausel sind jetzt nur noch diejenigen Zeilen der Tabelle *adresse* in der Ergebnistabelle enthalten, die nicht über Schlüsselwerte mit Zeilen der Tabelle *auszubildender* verbunden werden konnten.

10.2.4 Zeilen mit einem FULL OUTER JOIN verbinden

Ein **FULL OUTER JOIN** verbindet die Zeilen aus zwei Tabellen, die über übereinstimmende Schlüsselwerte verfügen.

Mit einem **FULL OUTER JOIN** werden auch die Zeilen ermittelt, die über keine übereinstimmenden Schlüsselwerte der Tabellen verbunden werden können. Ein **FULL OUTER JOIN** ist also eine Mischung aus **INNER JOIN**, **RIGHT OUTER JOIN** und **LEFT OUTER JOIN**.

MySQL unterstützt keinen FULL OUTER JOIN

Die MySQL-Datenbank unterstützt keinen **FULL OUTER JOIN** gemäß dem SQL-Standard. Mit den Kenntnissen, die Sie bis jetzt erworben haben, ist es möglich, einen **FULL OUTER JOIN** zu simulieren, um das gleiche Ergebnis zu erhalten. Zum Simulieren eines **FULL OUTER JOIN**s werden Sie im Anschluss dieses Abschnitts die **UNION**-Klausel verwenden.

Falls Sie eine MySQL-Datenbank verwenden, sollten Sie diesen Abschnitt dennoch aufmerksam lesen, da ein **FULL OUTER JOIN** nun einmal zum Standard gehört und oft genutzt wird.

Um den **FULL OUTER JOIN** besser zu verstehen, betrachten wir wieder Abbildung 10.6. In der Tabelle *auszubildender* sind Zeilen enthalten, auf die nicht mit einem Fremdschlüsselwert aus der Tabelle *adresse* referenziert wird. In der Tabelle *adresse* wiederum sind Zeilen abgelegt, die keine Fremdschlüsselwerte enthalten. Somit kann hier kein Bezug aus der Tabelle *adresse* zur Tabelle *auszubildender* hergestellt werden. Das kennen Sie bereits aus den Beispielen zum **LEFT OUTER JOIN** und **RIGHT OUTER JOIN**. Ein **FULL OUTER JOIN** stellt eine Kombination aus diesen beiden dar. Wenn Sie also Tabellen mit einem **FULL OUTER JOIN** verbinden, erhalten Sie alle Zeilen mit übereinstimmenden Schlüsselwerten sowie die Zeilen der linken und der rechten Tabelle, in denen keine Referenzierung von Schlüsselwerten hergestellt werden kann.

In Listing 10.19 sehen Sie eine **SELECT**-Abfrage, die die Zeilen der Tabellen *auszubildender* und *adresse* über einen **FULL OUTER JOIN** verbindet:

```
SELECT name,vorname,plz,ort
FROM auszubildender FULL OUTER JOIN adresse
ON ausid=fk_ausid;
```

Listing 10.19 Mit einem FULL OUTER JOIN Zeilen aus Tabellen verbunden abfragen

In Tabelle 10.10 sehen Sie das Ergebnis der **SELECT**-Abfrage aus Listing 10.19. Dort finden Sie Zeilen, die über Schlüsselwerte verbunden wurden. Außerdem sehen Sie Zei-

len mit Auszubildenden ohne passende Adresse und gleichzeitig Adressen, denen keine Auszubildenden zugeordnet werden konnten. Für die jeweils fehlenden Werte erhalten wir wieder nicht definierte Werte (`NULL`).

name	vorname	plz	ort
Müller	Ralf	50827	Köln
Klein	Sabine	NULL	NULL
Lang	Peter	50127	Bonn
Berg	Frank	NULL	NULL
Erde	Sabine	50129	Bonn
Grün	Justus	NULL	NULL
NULL	NULL	50129	Bonn
NULL	NULL	50127	Bonn
NULL	NULL	50827	Köln

Tabelle 10.10 Ergebnistabelle für eine Abfrage mit einem FULL OUTER JOIN

In den beiden letzten Abschnitten haben wir uns jeweils die Frage gestellt, wie wir die Zeilen eines Auszubildenden ermitteln, die keinem Wohnort zugeordnet sind, oder wie wir die Wohnorte ermitteln, die keinem Auszubildenden zuzuordnen sind. Wir haben eine Antwort auf diese Fragen gefunden, indem wir die betreffenden Zeilen mit einer `WHERE`-Klausel in einer `SELECT`-Abfrage gefiltert haben. Prüfen Sie einfach die Zeilen, in denen keine passende Eltern- oder Kindzeile mit einem Schlüssel verbunden werden kann, auf Gleichheit mit einem `NULL`-Wert.

In Listing 10.20 sehen Sie eine Abfrage, in der die `WHERE`-Klausel jeweils eine Spalte aus den Tabellen *auszubildender* und *adresse* auf Gleichheit mit `NULL` prüft. Wir haben es hier mit zwei Bedingungen zu tun, die mit einem logischen `OR`-Operator verknüpft werden. Das heißt, dass die `WHERE`-Klausel erfüllt ist, wenn eine der Bedingungen erfüllt ist. Oder anders gesagt: Die Bedingung ist wahr, wenn einer der Spaltenwerte der Spalten *name* oder *plz* gleich `NULL` ist.

```
SELECT name,vorname,plz,ort
FROM auszubildender FULL OUTER JOIN adresse
ON ausid=fk_ausid
where name IS NULL OR plz IS NULL;
```

Listing 10.20 In einem FULL OUTER JOIN eine Spalte der linken und eine Spalte der rechten Tabelle in einer WHERE-Klausel auf NULL prüfen

Tabelle 10.11 zeigt Ihnen das Ergebnis der Abfrage aus Listing 10.20. Die ersten drei Zeilen enthalten ausschließlich die Auszubildenden, denen nicht über passende Schlüsselwerte aus der Tabelle *adresse* Adressen zugeordnet werden können. Die Zeilen 4 bis 6 hingegen enthalten die Adressen, die aufgrund fehlender Schlüsselbeziehungen keinem Auszubildenden in der Tabelle *auszubildender* zugeordnet werden können.

Mit der Ergänzung einer `WHERE`-Klausel, in der jeweils Spaltenwerte der an einer `JOIN`-Verbindung beteiligten Tabellen auf `NULL` geprüft werden, können Sie prüfen, wo keine Beziehungen zwischen den Zeilen der beteiligten Tabellen existieren.

name	vorname	plz	ort
Klein	Sabine	NULL	NULL
Berg	Frank	NULL	NULL
Grün	Justus	NULL	NULL
NULL	NULL	50129	Bonn
NULL	NULL	50127	Bonn
NULL	NULL	50827	Köln

Tabelle 10.11 Ergebnistabelle für eine FULL OUTER JOIN-Abfrage, die mit einer WHERE-Klausel versehen wurde und prüft, ob eine Spalte je Tabelle gleich NULL ist

10.2.5 Einen FULL OUTER JOIN unter MySQL nachbilden

Dieser Abschnitt ist den Lesern gewidmet, die eine MySQL-Datenbank verwenden, damit sie einen `FULL OUTER JOIN` mithilfe einer `UNION`-Klausel nachbilden können. In Kapitel 5, »Mengenoperationen anwenden«, haben Sie gelernt, wie Sie Schnittmengen, Vereinigungsmengen und Differenzmengen von Zeilen bilden. Dieses Wissen werden wir uns jetzt zunutze machen, um einen `FULL OUTER JOIN` auf einer MySQL-Datenbank nachzubilden. Die in Abschnitt 10.2.4 dargestellten Beispiele werden wir in der gleichen Reihenfolge in unserer alternativen Lösung mit der `UNION`-Klausel nutzen, um einen `FULL OUTER JOIN` zu realisieren.

Betrachten Sie hierzu Listing 10.21. Es entspricht Listing 10.19. In dieser `SELECT`-Abfrage haben wir die Zeilen der Tabellen *auszubildender* und *adresse* über einen `FULL OUTER JOIN` verbunden.

```
SELECT name,vorname,plz,ort
FROM auszubildender FULL OUTER JOIN adresse
ON ausid=fk_ausid;
```

Listing 10.21 Mit einem FULL OUTER JOIN Zeilen aus Tabellen verbunden abfragen

Um einen **FULL OUTER JOIN** nachzubilden, betrachten Sie die drei **SELECT**-Abfragen in Listing 10.22, die jeweils einen **INNER JOIN**, einen **LEFT OUTER JOIN** und einen **RIGHT OUTER JOIN** realisieren:

```
SELECT name,vorname,plz,ort
FROM auszubildender INNER JOIN adresse
ON ausid=fk_ausid;

SELECT name,vorname,plz,ort
FROM auszubildender LEFT OUTER JOIN adresse
ON ausid=fk_ausid;

SELECT name,vorname,plz,ort
FROM auszubildender RIGHT OUTER JOIN adresse
ON ausid=fk_ausid;
```

Listing 10.22 INNER JOIN, LEFT OUTER JOIN und RIGHT OUTER JOIN

Die erste Abfrage aus Listing 10.22 verbindet die Zeilen der Tabellen *auszubildender* und *adresse*. Da es sich um einen **INNER JOIN** handelt, werden nur die Zeilen verbunden, die über gleiche Schlüsselwerte verfügen.

Die zweite Abfrage aus Listing 10.22 verbindet ebenfalls die Zeilen der Tabellen *auszubildender* und *adresse* miteinander. Hier werden die Zeilen über einen **LEFT OUTER JOIN** verbunden. Das bedeutet, dass alle Zeilen verbunden werden, die über gleiche Schlüsselwerte verfügen, und es werden die Zeilen aus der linken Tabelle mit aufgenommen, für die in der rechten Tabelle keine passenden Schlüsselwerte vorhanden sind.

Die dritte Abfrage aus Listing 10.22 verbindet die Zeilen der Tabellen *auszubildender* und *adresse* über einen **RIGHT OUTER JOIN**. Hier werden also alle Zeilen der rechten Tabelle mit allen Zeilen der linken Tabelle verbunden, die über gleiche Schlüsselwerte verfügen. Außerdem werden die Zeilen der rechten Tabelle mit ausgegeben, in denen keine Fremdschlüsselwerte festgelegt wurden.

Die drei **SELECT**-Abfragen haben eines gemeinsam: Sie fragen die Spalten *name*, *vorname*, *plz* und *ort* ab. Erinnern Sie sich noch an die **UNION**-Klausel aus Abschnitt 5.2? Dort haben Sie gelernt, dass für eine Vereinigung von Zeilen von mehreren Tabellen die Spaltenanzahl der einzelnen Abfragen identisch sein muss. Außerdem müssen die Datentypen in der Sequenz der Spaltenangabe identisch sein. Diese Voraussetzung ist hier gegeben.

Sehen wir uns dennoch noch einmal die Ergebnistabellen der drei **SELECT**-Abfragen an, damit Sie der Lösung auch vertrauen können.

Die erste Abfrage liefert, wie in Tabelle 10.12 gezeigt, die Ergebniszeilen, die aus einem **INNER JOIN** für die Zeilen der Tabellen *auszubildender* und *adresse* resultieren:

name	vorname	plz	ort
Müller	Ralf	50827	Köln
Lang	Peter	50127	Bonn
Erde	Sabine	50129	Bonn

Tabelle 10.12 Ergebnistabelle für eine Abfrage mit einem INNER JOIN

Die zweite Abfrage liefert, wie in Tabelle 10.13 gezeigt, die Ergebniszeilen, die aus einem **LEFT OUTER JOIN** für die Zeilen der Tabellen *auszubildender* und *adresse* resultieren:

name	vorname	plz	ort
Müller	Ralf	50827	Köln
Klein	Sabine	NULL	NULL
Lang	Peter	50127	Bonn
Berg	Frank	NULL	NULL
Erde	Sabine	50129	Bonn
Grün	Justus	NULL	NULL

Tabelle 10.13 Ergebnistabelle für eine Abfrage mit einem LEFT OUTER JOIN

Die dritte Abfrage liefert Ihnen, wie in Tabelle 10.14 gezeigt, die Ergebniszeilen, die aus einem **RIGHT OUTER JOIN** für die Zeilen der Tabellen *auszubildender* und *adresse* resultieren:

name	vorname	plz	ort
Müller	Ralf	50827	Köln
Lang	Peter	50127	Bonn
Erde	Sabine	50129	Bonn
NULL	NULL	50827	Köln
NULL	NULL	50127	Bonn
NULL	NULL	50129	Bonn

Tabelle 10.14 Ergebnistabelle für eine Abfrage mit einem RIGHT OUTER JOIN

Die Übersicht der drei Ergebnistabellen zeigt Ihnen, dass wir mit einer Vereinigung der Ergebniszeilen aus den drei Ergebnistabellen einen **FULL OUTER JOIN** in seiner Funktionalität nachbilden können.

In Listing 10.23 sehen Sie die Vereinigung der Ergebniszeilen der drei Abfragen, die mit einer **UNION**-Klausel realisiert wird. Es werden einfach die Zeilen der Ergebnistabellen der drei **SELECT**-Abfragen vereinigt, die jeweils mit einer **INNER JOIN**-Klausel, einer **LEFT OUTER JOIN**-Klausel und einer **RIGHT OUTER JOIN**-Klausel ausgestattet sind.

```
SELECT name,vorname,plz,ort
FROM auszubildender INNER JOIN adresse
ON ausid=fk_ausid
UNION
SELECT name,vorname,plz,ort
FROM auszubildender LEFT OUTER JOIN adresse
ON ausid=fk_ausid
UNION
SELECT name,vorname,plz,ort
FROM auszubildender RIGHT OUTER JOIN adresse
ON ausid=fk_ausid;
```

Listing 10.23 Einen FULL OUTER JOIN mit UNION nachbilden

Wie bei einem **FULL OUTER JOIN** erhalten Sie als Ergebnis eine Tabelle, die die Zeilen enthält, die über Schlüsselwerte verbunden werden konnten. Die Zeilen der linken Tabelle, die mit keinem Schlüsselwert der rechten Tabelle übereinstimmen, und die Zeilen der rechten Tabelle, für die kein Fremdschlüsselwert festgelegt wurde, sind ebenfalls Bestandteil des Ergebnisses.

Die Zeilen, die ausschließlich in der linken oder rechten Tabelle enthalten sind, werden von der Datenbank automatisch mit **NULL**-Werten aufgefüllt. Das Auffüllen mit **NULL**-Werten ist dem **LEFT OUTER JOIN** und dem **RIGHT OUTER JOIN** geschuldet, die wir neben den Ergebniszeilen des **INNER JOIN**s in der Vereinigung der Ergebniszeilen verwendet haben.

Das Ergebnis sehen Sie in Tabelle 10.15:

name	vorname	plz	ort
Müller	Ralf	50827	Köln
Lang	Peter	50127	Bonn
Erde	Sabine	50129	Bonn
Klein	Sabine	NULL	NULL

Tabelle 10.15 Einen FULL OUTER JOIN mit UNION nachbilden

name	vorname	plz	ort
Berg	Frank	NULL	NULL
Grün	Justus	NULL	NULL
NULL	NULL	50827	Köln
NULL	NULL	50127	Bonn
NULL	NULL	50129	Bonn

Tabelle 10.15 Einen FULL OUTER JOIN mit UNION nachbilden (Forts.)

Als Nächstes werden wir, wie in den anderen Beispielen, mit einer **WHERE**-Klausel filtern, welche Zeilen der Tabellen *auszubildender* und *adresse* nicht über Schlüsselbeziehungen verbunden werden können. An dieser Stelle können wir den **INNER JOIN** aus der **UNION**-Operation weglassen, da uns in diesem Beispiel ausschließlich die Zeilen interessieren, die nicht verbunden werden können.

In Listing 10.24 sehen Sie, dass hinter den einzelnen **SELECT**-Anweisungen mit einer **WHERE**-Klausel jeweils eine Spalte der rechten bzw. linken Tabelle auf **NULL** geprüft wird, um nur die Zeilen zu ermitteln, die nicht über Schlüsselwerte verbunden werden können:

```
SELECT name,vorname,plz,ort
FROM auszubildender LEFT OUTER JOIN adresse
ON ausid=fk_ausid
WHERE plz IS NULL
UNION
SELECT name,vorname,plz,ort
FROM auszubildender RIGHT OUTER JOIN adresse
ON ausid=fk_ausid
WHERE name IS NULL;
```

Listing 10.24 Die Zeilen ermitteln, die nicht über einen JOIN verbunden werden konnten

Tabelle 10.16 zeigt Ihnen das Ergebnis der Abfrage aus Listing 10.24. Die ersten drei Zeilen enthalten – wie im Beispiel aus Abschnitt 10.2.4 – wieder ausschließlich die Auszubildenden, denen nicht über passende Schlüsselwerte aus der Tabelle *adresse* Adressen zugeordnet werden können. Die Zeilen 4 bis 6 hingegen enthalten wieder die Adressen, die aufgrund fehlender Schlüsselbeziehungen keinem Auszubildenden in der Tabelle *auszubildender* zugeordnet werden können.

name	vorname	plz	ort
Klein	Sabine	NULL	NULL
Berg	Frank	NULL	NULL
Grün	Justus	NULL	NULL
NULL	NULL	50827	Köln
NULL	NULL	50127	Bonn
NULL	NULL	50129	Bonn

Tabelle 10.16 Nur die Zeilen ermitteln, die nicht verbunden abgefragt werden konnten

10.2.6 Zeilen mit einem CROSS JOIN verbinden

Ein **CROSS JOIN** verbindet völlig unabhängig von Schlüsselbeziehungen zwischen Eltern- und Kindtabellen die Zeilen einer Tabelle mit allen Zeilen einer anderen Tabelle. Betrachten Sie eine Zeile der linken Tabelle, so wird diese Zeile mit jeder Zeile der rechten Tabelle verbunden.

Mathematisch betrachtet handelt es sich hierbei um ein kartesisches Produkt. Sie können **CROSS JOIN**s verwenden, um Testdaten zu generieren oder um alle Zeilen von zwei Tabellen miteinander verbunden auszugeben. Wenn Sie Testdaten generieren wollen, so reicht eine Tabelle mit 20.000 Zeilen aus, die Sie zum Beispiel mit einer Tabelle, die 100 Zeilen enthält, verbunden abfragen. So generieren Sie eine Ergebnistabelle mit Testdaten, die 2.000.000 Zeilen enthält.

Um Ihnen die Funktion eines **CROSS JOIN**s möglichst einfach zu erklären, bleiben wir aber bei den Auszubildenden, denen Adressen zugeordnet sind.

In Abbildung 10.7 sehen Sie, dass die erste Zeile der Tabelle *adresse* mit allen Zeilen der Tabelle *auszubildender* verbunden wird. Somit gilt auch, dass jede Zeile der rechten Tabelle *auszubildender* mit allen Zeilen der Tabelle *adresse* verbunden ist.

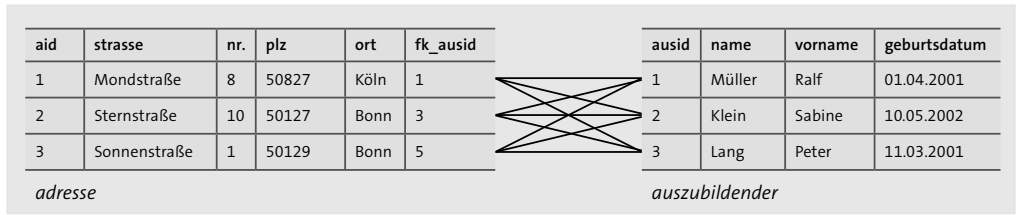


Abbildung 10.7 Zeilen von Tabellen, die über einen CROSS JOIN verbunden abgefragt werden

Unser Ziel ist es, jeden Auszubildenden mit allen Wohnorten sowie alle Wohnorte mit jedem Auszubildenden zu verbinden. Der praktische Nutzen dieser Verbindung

ist zwar nicht gerade offensichtlich, aber als Erklärung eignet sich das in Abbildung 10.7 gezeigte Beispiel gut, um Sie mit dem **CROSS JOIN** vertraut zu machen.

Um eine **SELECT**-Abfrage über zwei Tabellen mit einem **CROSS JOIN** auszustatten, notieren Sie zwischen den Tabellen *auszubildender* und *adresse* die Schlüsselwörter **CROSS JOIN**. Das war es schon, da ein **CROSS JOIN** ja lediglich alle Zeilen einer Tabelle mit allen Zeilen einer anderen Tabelle verbindet, und das auch umgekehrt. Es wird also keine Bedingung (**ON**-Klausel mit Bedingung) benötigt, die Schlüsselwerte auf Gleichheit prüft. In dieser **SELECT**-Abfrage wurde noch die Spalte *strasse* mit in die Spaltenauswahlliste aufgenommen. So können Sie leichter sehen, dass tatsächlich alle Zeilen einer Tabelle mit allen Zeilen einer anderen Tabelle verbunden werden und umgekehrt.

```
SELECT name,vorname,plz,ort,strasse
FROM auszubildender CROSS JOIN adresse;
```

Listing 10.25 Zeilen aus zwei Tabellen mit einem CROSS JOIN verbunden abfragen

Tabelle 10.17 zeigt Ihnen das Ergebnis der **SELECT**-Abfrage aus Listing 10.25.

name	vorname	plz	ort	strasse
Müller	Ralf	50827	Köln	Mondstraße
Klein	Sabine	50827	Köln	Mondstraße
Lang	Peter	50827	Köln	Mondstraße
Berg	Frank	50827	Köln	Mondstraße
Erde	Sabine	50827	Köln	Mondstraße
Grün	Justus	50827	Köln	Mondstraße
Müller	Ralf	50127	Bonn	Sternstraße
Klein	Sabine	50127	Bonn	Sternstraße
Lang	Peter	50127	Bonn	Sternstraße
Berg	Frank	50127	Bonn	Sternstraße

Tabelle 10.17 Auszug aus der Ergebnistabelle für eine SELECT-Abfrage mit einem CROSS JOIN

Sie sehen, dass jede Zeile der Tabelle *auszubildender* mit jeder Zeile der Tabelle *adresse* ausgegeben wird. Wenn Sie die Abfrage nachvollziehen, werden Sie sehen, dass aus den sechs Zeilen der Tabelle *auszubildender*, in denen die Auszubildenden

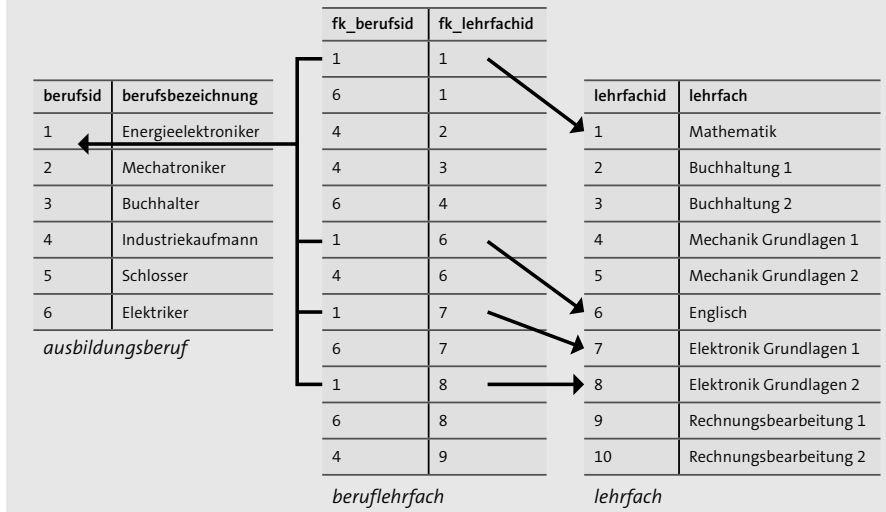
hinterlegt werden, und den sechs Zeilen der Tabelle *adresse*, in denen Wohnorte gespeichert sind, dank einem **CROSS JOIN** eine Ergebnistabelle mit 36 Zeilen wird.

In Tabelle 10.17 ist lediglich ein Auszug aus der Ergebnistabelle zu sehen. Um zu prüfen, ob es sich tatsächlich um 36 Zeilen handelt, sollten Sie das Beispiel praktisch nachvollziehen.

10.2.7 Zeilen von drei Tabellen mit einem INNER JOIN verbinden

Sie erinnern sich bestimmt, dass ich Ihnen in Abschnitt 7.4 die Ausbildungsdatenbank als Modell in der UML-Notation vorgestellt habe. Hier stehen die Tabellen *ausbildungsberuf*, *beruflehrfach* und *lehrfach* in Beziehung zueinander.

Die Werte der Fremdschlüsselspalten *fk_berufsid* und *fk_lehrfachid* aus der Tabelle *beruflehrfach* zeigen jeweils auf Werte der Primärschlüsselspalten *berufsid* und *lehrfachid* der Tabellen *ausbildungsberuf* und *lehrfach*. Somit sind die optimalen Voraussetzungen gegeben, um mit einem **INNER JOIN** Zeilen aus drei Tabellen zu verbinden und abzufragen.



berufsid	berufsbezeichnung
1	Energieelektroniker
2	Mechatroniker
3	Buchhalter
4	Industriekaufmann
5	Schlosser
6	Elektriker

ausbildungsberuf

fk_berufsid	fk_lehrfachid
1	1
6	1
4	2
4	3
6	4
1	6
4	6
1	7
6	7
1	8
6	8
4	9

beruflehrfach

lehrfachid	lehrfach
1	Mathematik
2	Buchhaltung 1
3	Buchhaltung 2
4	Mechanik Grundlagen 1
5	Mechanik Grundlagen 2
6	Englisch
7	Elektronik Grundlagen 1
8	Elektronik Grundlagen 2
9	Rechnungsbearbeitung 1
10	Rechnungsbearbeitung 2

lehrfach

Abbildung 10.8 Drei Tabellen, die zueinander in Beziehung stehen

In Listing 10.26 sehen Sie eine **SELECT**-Abfrage, in der ein **INNER JOIN** über die drei Tabellen *ausbildungsberuf*, *beruflehrfach* und *lehrfach* realisiert wird.

```
SELECT berufsbezeichnung, lehrfach
FROM ausbildungsberuf
INNER JOIN beruflehrfach
ON berufsid=fk_berufsid
```



```
INNER JOIN lehrfach
ON fk_lehrfachid=lehrfachid;
```

Listing 10.26 Zeilen von drei Tabellen mit einem INNER JOIN verbunden abfragen

Vom Grundsatz her habe ich den **INNER JOIN** bereits in Abschnitt 10.2.1 behandelt. Der Unterschied in Listing 10.26 besteht zunächst darin, dass hinter der ersten **ON**-Klausel ein weiterer **INNER JOIN** angegeben wird, der die Tabelle *lehrfach* verbindet. In der Bedingung der **ON**-Klausel wird auf Gleichheit zwischen den Spaltenwerten der Fremdschlüsselspalte *fk_lehrfach* und der Primärschlüsselspalte *lehrfachid* geprüft.

Zunächst wird also mit der ersten **ON**-Klausel auf Gleichheit zwischen den Spaltenwerten der Spalten *berufsid* und *fk_berufsid* geprüft und somit eine Verbindung zwischen den Tabellen *ausbildungsberuf* und *beruflehrfach* geschaffen.

Dann wird in der zweiten **ON**-Klausel auf Gleichheit zwischen den Spaltenwerten der Spalten *fk_lehrfach* und *lehrfachid* geprüft und somit eine weitere Verbindung zwischen den Tabellen *beruflehrfach* und *lehrfach* geschaffen.

Mehrere Tabellen verbinden

Sie können natürlich auch mehr als drei Tabellen mit einem **INNER JOIN** verbunden abfragen. Sie müssen lediglich hinter der letzten **ON**-Klausel eine weitere **INNER JOIN**-Klausel mit der dazugehörigen **ON**-Klausel anfügen.

In Tabelle 10.18 sehen Sie das Ergebnis der **SELECT**-Abfrage aus Listing 10.26.

berufsbezeichnung	lehrfach
Energieelektroniker	Mathematik
Energieelektroniker	Englisch
Energieelektroniker	Elektronik Grundlagen 1
Energieelektroniker	Elektronik Grundlagen 2
Industriekaufmann	Buchhaltung 1
Industriekaufmann	Buchhaltung 2
Industriekaufmann	Englisch
Industriekaufmann	Rechnungsbearbeitung 1

Tabelle 10.18 Ergebnistabelle für eine verbundene Abfrage aus drei Tabellen

Die Abfrage liefert uns verbunden über die Schlüsseltabelle *beruflehrfach* die Berufsbezeichnungen und die jeweiligen Lehrfächer als Ergebnistabelle zurück.

10.2.8 Spalten in einem JOIN über Tabellennamen referenzieren

Bisher haben Sie Zeilen von Tabellen verbunden, deren Spaltenbezeichnungen unterschiedlich sind. Stellen Sie sich vor, Sie formulieren einen **INNER JOIN** über zwei Tabellen, in denen Spalten mit gleichen Spaltennamen vorkommen. Wenn gleichnamige Spaltennamen in der Spaltenauswahlliste einer **SELECT**-Anweisung enthalten sind, kann nicht zwischen den Spalten unterschieden werden. Wenn gleichnamige Spalten in zwei oder mehreren Tabellen vorkommen, kann die Datenbank also nicht unterscheiden, welche Spalte aus welcher Tabelle ausgewählt werden soll.

Das Problem ist allerdings leicht zu lösen: Schreiben Sie die jeweiligen Spaltennamen in der Spaltenauswahlliste in der Notation *tabellenname.spaltenname*. Zuerst notieren Sie also die Tabelle, gefolgt von einem Punkt, und schließlich dem Spaltennamen.

Sehen wir uns als Nächstes ein Beispiel an. In Abbildung 10.9 sehen Sie einen Ausschnitt unseres Datenmodells für eine Ausbildungsdatenbank in der UML-Notation.

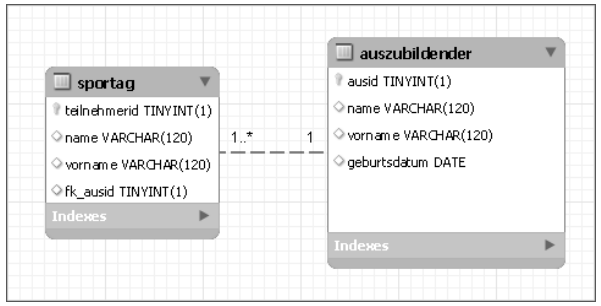


Abbildung 10.9 Tabellen, die identische Spaltennamen enthalten

Das Datenmodell wurde um die Tabelle *sportag* (Sport-Arbeitsgruppe) erweitert. In der Tabelle sind die Spalten *teilnehmerid*, *name*, *vorname* und *fk_ausid* enthalten. Die Spalte *fk_ausid* repräsentiert hier wieder die Fremdschlüsselspalte. Die Spaltennamen *name* und *vorname* kommen in den Tabellen *auszubildender* und *sportag* vor.

Um das Beispiel nachvollziehen zu können, verwenden wir die **CREATE TABLE**-Anweisung aus Listing 10.27, die die Tabelle *sportag* in der *ausbildungsdatenbank* anlegt:

```
CREATE TABLE sportag (
  teilnehmerid TINYINT PRIMARY KEY NOT NULL, /* PostgreSQL SMALLINT */
  name VARCHAR(120),
  vorname VARCHAR(120),
  fk_ausid TINYINT, /* PostgreSQL SMALLINT */
  CONSTRAINT fk_ausid_auszubildender
    FOREIGN KEY (fk_ausid)
      REFERENCES auszubildender (ausid)
);
```

Listing 10.27 CREATE TABLE-Anweisung für die Tabelle »sportag«

Listing 10.28 enthält zwei **INSERT**-Anweisungen, die benötigt werden, um zwei Zeilen in die Tabelle *sportag* einzufügen:

```
INSERT INTO sportag (teilnehmerid,name,vorname,fk_ausid)
  VALUES (1,'Müller','Ralf',1);

INSERT INTO sportag (teilnehmerid,name,vorname,fk_ausid)
  VALUES (2,'Klein','Sabine',2);
```

Listing 10.28 Zeilen in die Tabelle »sportag« einfügen

In Abbildung 10.10 sehen Sie die Tabellen *sportag* und *auszubildender*. Sie enthalten beide die Spaltennamen *name* und *vorname*.

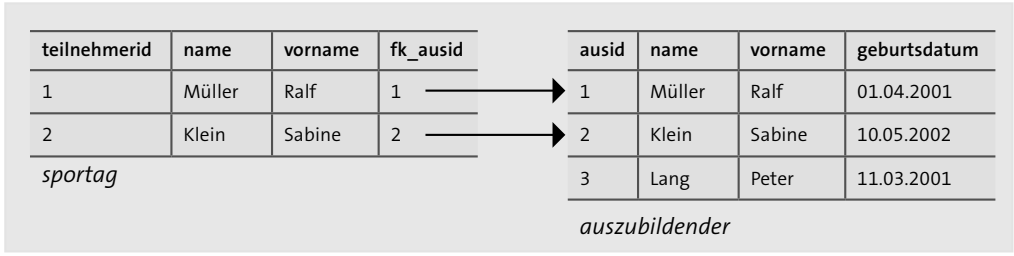


Abbildung 10.10 Tabellen mit gleichen Spaltennamen

Als Nächstes betrachten wir eine **SELECT**-Abfrage mit einem **INNER JOIN**, in dem die Spalten *name* und *vorname* abgefragt werden sollen. Außerdem soll aus der Tabelle *auszubildender* die Spalte *geburtsdatum* abgefragt werden.

```
SELECT name,vorname,geburtsdatum
FROM auszubildender INNER JOIN sportag
ON ausid=fk_ausid;
```

Listing 10.29 Mit einem INNER JOIN die Tabellen »auszubildender« und »sportag« verbunden abfragen

Die **SELECT**-Abfrage aus Listing 10.29 führt zu einem Fehler, weil das Datenbanksystem die Spalten *name* und *vorname* nicht eindeutig einer Tabelle zuordnen kann. Die MySQL Datenbank reagiert auf mehrfach gleich vorkommende Spaltennamen mit der Fehlermeldung:

```
Error Code: 1052. Column 'name' in field list is ambiguous
```

Die Lösung für dieses Problem sehen Sie in Listing 10.30. Dort werden sämtliche Spalten jeweils über einen Tabellennamen referenziert. Jetzt kann die Datenbank die jeweiligen Spalten den Tabellen zuordnen, und die Mehrdeutigkeit ist aufgehoben.

```
SELECT
  auszubildender.name,
  auszubildender.vorname,
  auszubildender.geburtsdatum
FROM auszubildender INNER JOIN sportag
ON auszubildender.ausid=sportag.fk_ausid;
```

Listing 10.30 Spalten über einen Tabellennamen referenzieren

Tabelle 10.19 zeigt das Ergebnis der Abfrage. Die **SELECT**-Abfrage konnte jetzt problemlos ausgeführt werden.

name	vorname	geburtsdatum
Müller	Ralf	2001-04-01
Klein	Sabine	2002-05-10

Tabelle 10.19 Ergebnistabelle für eine INNER JOIN-Abfrage mit mehrdeutigen Spaltennamen

In diesem Abschnitt haben Sie erfahren, wie Sie Spaltennamen aus zwei oder mehreren Tabellen über Tabellennamen referenzieren. Fällt Ihnen vielleicht auf, dass es noch einen anderen Weg gibt, das Problem der übereinstimmenden Bezeichnungen zu lösen?

Es wäre auch möglich gewesen, ohne die Spalten *name* und *vorname* in der Tabelle *sportag* auszukommen. Über die Fremdschlüsselbeziehung ist ein Zugriff auf die Spalten *name* und *vorname* in der Tabelle *auszubildender* ja bereits gegeben. Durch aufmerksames Nachdenken, welche Spalten überhaupt notwendig sind, hätte sich das Hindernis also auch umgehen lassen. Hier wollte ich Ihnen aber zeigen, wie Sie mit gleichen Spaltennamen umgehen, und deswegen habe ich dieses Beispiel gewählt.

10.2.9 Spalten in einem JOIN über Tabellenaliasse referenzieren

Wenn Sie Tabellen verbunden abfragen, können Sie hinter den Tabellennamen in der **SELECT**-Anweisung einen Tabellenalias notieren. Mit dem vergebenen Tabellenalias können Sie wiederum auf die Spalten der jeweiligen Tabelle referenzieren. Listing 10.31 ist funktionell identisch mit der **SELECT**-Abfrage aus Listing 10.30:

```
SELECT a.name,a.vorname,a.geburtsdatum
FROM auszubildender a INNER JOIN sportag s
ON a.ausid=s.fk_ausid;
```

Listing 10.31 Spalten über Tabellenaliasse referenzieren

In Listing 10.31 werden Tabellenaliasse verwendet, die direkt hinter dem Tabellennamen notiert werden. Der Tabelle *auszubildender* ist der Tabellenalias *a* zugeordnet, der Tabelle *sportag* wiederum der Tabellenalias *s*.

Die festgelegten Tabellenaliasse *a* und *s* werden in der Spaltenauswahlliste der **SELECT**-Anweisung und in der **ON**-Klausel verwendet. Wenn Sie Spalten über einen Tabellenalias referenzieren, ersparen Sie sich eine Menge Schreibarbeit, und die Abfrage wird übersichtlicher. In der Regel werden Spalten aus Tabellen, die mit einem **JOIN** verbunden sind, über Tabellenaliasse referenziert.

Das Ergebnis ist exakt identisch mit dem Ergebnis, das aus Listing 10.31 resultiert. Aus dem Grund verzichte ich hier auf eine Ergebnistabelle.

In Kapitel 2, »Los geht's: Die Grundfunktionen der Tabellenabfrage (SELECT)«, haben Sie erfahren, wie Sie Spalten einen Spaltenalias zuordnen. In den Tabellen *auszubildender* und *sportag* sind jeweils die Spalten *name* und *vorname* enthalten. Zwei gleichnamige Spalten, die aus einem **JOIN** hervorgehen, sollten mit Spaltenaliassen abgefragt werden, um die Spalten auseinanderzuhalten, die ja aus unterschiedlichen Tabellen kommen. Sehen wir uns als Nächstes ein Beispiel an, in dem keine Spaltenaliasse verwendet und die gleichnamigen Spalten aus den Tabellen *auszubildender* und *sportag* abgefragt werden:

```
SELECT a.name,a.vorname,a.geburtsdatum,s.name,s.vorname
FROM auszubildender a INNER JOIN sportag s
ON a.ausid=s.fk_ausid;
```

Listing 10.32 Tabellen mit gleichen Spaltennamen mit einem INNER JOIN abfragen

Tabelle 10.20 zeigt das Ergebnis der Abfrage aus Listing 10.32.

name	vorname	geburtsdatum	name	vorname
Müller	Ralf	2001-04-01	Müller	Ralf
Klein	Sabine	2002-05-10	Klein	Sabine

Tabelle 10.20 Ergebnistabelle mit gleichen Spaltennamen

Sie sehen, dass Sie die jeweils doppelt vorkommenden Spalten *name* und *vorname* anhand des Namens keiner Tabelle zuordnen können. Hier helfen uns die Spaltenaliasse, die Sie in Abschnitt 2.7 kennengelernt haben.

In Listing 10.33 sehen Sie eine **SELECT**-Abfrage, in der die Spalten der Spaltenauswahlliste jeweils mit einem Spaltenalias versehen wurden:

```
SELECT
a.name AS aus_name,
a.vorname AS aus_vorname,
```

```
a.geburtsdatum AS aus_gebdatum,
s.name AS spag_name,
s.vorname AS spag_vorname
FROM auszubildender a INNER JOIN sportag s
ON a.ausid=s.fk_ausid;
```

Listing 10.33 Doppelt vorkommende Spalten in einem JOIN mit Spaltenaliassen versehen

Als Resultat erhalten wir eine Ergebnistabelle, in der wir die Spalten eindeutig den Tabellen zuordnen können, aus denen sie stammen.

aus_name	aus_vorname	aus_gebdatum	spag_name	spag_vorname
Müller	Ralf	2001-04-01	Müller	Ralf
Klein	Sabine	2002-05-10	Klein	Sabine

Tabelle 10.21 Ergebnistabelle mit Spaltenaliassen

10.2.10 Zeilen mit einem SELF JOIN verbinden

Bevor wir uns mit dem **SELF JOIN** im Detail befassen, müssen wir klären, wozu er verwendet wird. Mit einem **SELF JOIN** können Sie hierarchische Strukturen einer Tabelle abbilden, die über die entsprechenden Voraussetzungen verfügt. Um hierarchische Strukturen in einer Tabelle abzubilden, verbinden Sie die Zeilen einer Tabelle mit den Zeilen ebendieser einen Tabelle. Um dies zu realisieren, nutzen Sie eine Fremdschlüsselbeziehung. Der Unterschied ist der, dass der Fremdschlüssel nicht auf einen Primärschlüssel in einer anderen Tabelle verweist, sondern auf den Primärschlüssel, der in der gleichen Tabelle definiert wurde wie der Fremdschlüssel. Sie nutzen also ausschließlich eine Tabelle, um ihre Zeilen wieder miteinander zu verbinden. Eine Tabelle stellt also die Eltern- und die Kindtabelle gleichzeitig dar.

Um einen **SELF JOIN** besser zu verstehen, ist es gut, die abzufragende Tabelle, deren Zeilen mit einem **JOIN** verbunden werden sollen, wie zwei eigenständige Tabellen zu betrachten. Wir stellen uns also vor, dass es sich um zwei verschiedene Tabellen handelt. In der ersten Tabelle finden wir eine Primärschlüsselspalte vor. In der zweiten Tabelle befindet sich eine Fremdschlüsselspalte, die wiederum auf die Primärschlüsselspalte der ersten Tabelle verweist.

Der SELF JOIN ist keine SQL-Klausel

Bitte beachten Sie, dass der SELF JOIN keine SQL-Klausel im klassischen Sinne ist. Es handelt sich lediglich um eine Bezeichnung für einen JOIN, der ausschließlich auf eine Tabelle angewendet wird. Einen SELF JOIN können Sie mit einem **INNER JOIN**, einem **LEFT OUTER JOIN**, einem **RIGHT OUTER JOIN** und mit einem **FULL OUTER JOIN** realisieren.

In Abbildung 10.11 sehen Sie so eine Beziehung, die wir uns vorgestellt haben.

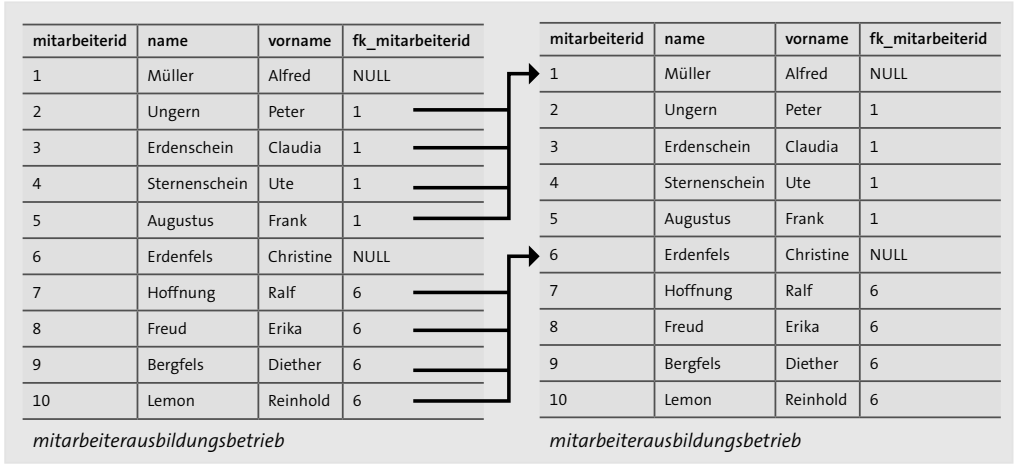


Abbildung 10.11 Eine Tabelle mit Zeilen, die auf Zeilen in der gleichen Tabelle referenzieren

In der linken Tabelle sehen Sie die Primärschlüsselspalte *mitarbeiterid*. In der rechten Tabelle sehen Sie die Fremdschlüsselspalte *fk_mitarbeiterid*. Die Fremdschlüsselwerte der Spalte *fk_mitarbeiterid* der rechten Tabelle verweisen wiederum auf die Primärschlüsselwerte der Spalte *mitarbeiterid*.

Mit diesen Erkenntnissen können wir einen **INNER JOIN** verwenden, der ermittelt, welche Mitarbeiter welchen Vorgesetzten untergeordnet sind. Den Mitarbeitern Alfred Müller und Christine Erdenfels sind in diesem Beispiel gleichzeitig die Rollen des Vorgesetzten und Inhabers zugeordnet, da keine Fremdschlüsselwerte (**NULL**) für diese Mitarbeiter existieren.

Als Nächstes sehen wir uns Listing 10.34 an:

```
SELECT
  ma1.mitarbeiterid AS vorgesetzterid,
  ma1.name AS vorgesetzername,
  ma2.name AS mitarbeitername,
  ma2.mitarbeiterid AS mitarbeiterid
FROM mitarbeiterausbildungsbetrieb ma1 INNER JOIN
  mitarbeiterausbildungsbetrieb ma2
ON ma1.mitarbeiterid=ma2.fk_mitarbeiterid;
```

Listing 10.34 Eine Tabelle mit sich selbst verbunden abfragen

Hier werden mit einem **INNER JOIN** die Zeilen einer Tabelle mit sich selbst verbunden abgefragt. Daraus ergibt sich die Notwendigkeit, Tabellenaliasse zu verwenden. Ohne

die Tabellenaliasse könnte die Datenbank, wie im letzten Abschnitt beschrieben, nicht unterscheiden, welche Spalten zu welcher Tabelle gehören. Darum wurden in diesem Beispiel Spaltenaliasse vergeben.

Sonst entspricht der **INNER JOIN** den gleichen Regeln, die Sie bereits kennengelernt haben.

In Tabelle 10.22 sehen Sie das Ergebnis der Abfrage.

vorgesetzterid	vorgesetzternname	mitarbeitername	mitarbeiterid
1	Müller	Ungern	2
1	Müller	Erdenschein	3
1	Müller	Sternenschein	4
1	Müller	Augustus	5
6	Erdenfels	Hoffnung	7
6	Erdenfels	Freud	8
6	Erdenfels	Bergfels	9
6	Erdenfels	Lemon	10

Tabelle 10.22 Ergebnistabelle für einen INNER JOIN, der Zeilen einer Tabelle miteinander verbindet

Die Zeilen der Vorgesetzten (Tabellenalias *ma1*) wurden immer dann mit den Zeilen der Mitarbeiter (Tabellenalias *ma2*) verbunden, wenn die Fremdschlüsselwerte, die auf die Mitarbeiter zeigen, gleich den Primärschlüsselwerten der Vorgesetzten sind.

10.2.11 Zeilen mit einem INNER JOIN ohne Schlüsselvergleiche verbinden

Bisher haben Sie **JOIN**-Typen in einer **SELECT**-Abfrage verwendet, um die Zeilen von zwei oder mehreren Tabellen miteinander zu verbinden. Die **JOIN**s, die wir bisher formuliert haben, wurden stets über gültige Schlüsselbeziehungen ausgewertet, die bereits in der **CREATE TABLE**-Anweisung festgelegt waren. Es war also immer eine Bedingung vorhanden, die in der **ON**-Klausel genutzt werden konnte, um Spaltenwerte zu vergleichen.

In diesem Abschnitt zeige ich Ihnen, dass Sie einen beliebigen **JOIN**-Typ unabhängig von der Existenz von Fremdschlüsselbeziehungen verwenden können. In der **ON**-

Klausel können beliebige Spalten aus zwei Tabellen verglichen werden. Das funktioniert allerdings nur, wenn der Datentyp der zu vergleichenden Spaltenwerte gleich ist oder implizit von der Datenbank konvertiert werden kann. In Abschnitt 4.3 habe ich Ihnen den Unterschied zwischen einer expliziten und einer impliziten Typkonvertierung erklärt. Jetzt ist für Sie wichtig zu wissen, dass in der Bedingung der **ON**-Klausel Spaltenwerte des gleichen Typs ausgewertet werden.

Sehen wir uns als Erstes das Beispiel in Abbildung 10.12 an.

Abbildung 10.12 Verbindungen zwischen Zeilen mit Nicht-Schlüsselspalten realisieren

Hier sehen Sie die bekannten Tabellen *auszubildender* und *sportag*, in denen jeweils die Spalte *name* enthalten ist. Die Datentypen sind in beiden Tabellen gleich, nämlich **VARCHAR(120)**. Somit hindert uns nichts daran, diese beiden Spalten wie Schlüsselspalten in einer Bedingung einer **ON**-Klausel zu verwenden. Wir könnten also einen **INNER JOIN** formulieren, um die Spalte *name*, die hier in beiden Tabellen vorhanden ist, in einer Bedingung auszuwerten. Auch hier werden die Zeilen der Tabellen nur dann verbunden ausgegeben, wenn die Bedingung in der **ON**-Klausel erfüllt ist.

Kommen wir zu einem konkreten Beispiel:

```
SELECT a.name, a.vorname, a.geburtsdatum, s.teilnehmerid
FROM auszubildender a INNER JOIN sportag s
ON a.name=s.name;
```

Listing 10.35 Ein INNER JOIN mit einer Bedingung, die Spalten in der ON-Klausel auswertet, die keine Schlüsselspalten sind

In Listing 10.35 ist eine **SELECT**-Abfrage mit einem **INNER JOIN** zu sehen, die in der **ON**-Klausel die Spalte *name* der beiden Tabellen *auszubildender* und *sportag* auf Gleichheit prüft.

In Tabelle 10.23 sehen Sie das Ergebnis der **SELECT**-Abfrage aus Listing 10.35. Die Zeilen der Tabellen *auszubildender* und *sportag* werden immer dann miteinander verbunden und in die Ergebnistabelle mit aufgenommen, wenn die Werte der Spalte *name* der Bedingung in der **ON**-Klausel entsprechen.

name	vorname	geburtsdatum	teilnehmerid
Müller	Ralf	2001-04-01	1
Klein	Sabine	2002-05-10	2

Tabelle 10.23 Ergebnis eines INNER JOINs, der ohne Schlüsselspalten gebildet wurde

[*]

Zusammenfassung: Zeilen, die in Beziehung stehen, abfragen

Mit folgenden **JOIN**-Typen können Tabellen verbunden abgefragt werden:

- **INNER JOIN** – verbindet ausschließlich Zeilen mit gültigen Schlüsselwerten.
- **LEFT OUTER JOIN** – verbindet Zeilen mit gültigen Schlüsselwerten und gibt Zeilen der linken Tabelle aus, die nicht verbunden abgefragt werden können.
- **RIGHT OUTER JOIN** – verbindet Zeilen mit gültigen Schlüsselwerten und gibt Zeilen der rechten Tabelle aus, die nicht verbunden abgefragt werden können.
- **FULL OUTER JOIN** – verbindet Zeilen mit gültigen Schlüsselwerten und gibt die Zeilen der linken und rechten Tabelle aus, die nicht über Schlüsselwerte verbunden werden können.
- **CROSS JOIN** – verbindet jede Zeile einer Tabelle mit jeder Zeile einer anderen Tabelle und umgekehrt.
- **SELF JOIN** – verbindet eine Zeile einer Tabelle mit Zeilen der gleichen Tabelle.

Die Verbindung zwischen den Zeilen wird (bis auf den **CROSS JOIN** mit einer Bedingung) stets über Primärschlüsselwerte und darauf referenzierende Fremdschlüsselwerte hergestellt.

Die hier aufgeführten **JOIN**-Typen (bis auf den **CROSS JOIN**) können nicht nur über Schlüsselbeziehungen realisiert werden. Solange die Datentypen der Spalten, die an einem **JOIN** beteiligt sind, identisch sind, kann mit einer Bedingung in einer **ON**-Klausel eine Verbindung hergestellt werden.

Auf einen Blick

Inhalt